القسم الأول والثاني والثالث والرابع

البرنامج الحاسوبي

الكلمات المفتاحية:

ترميز، مُترجم، لغة برمجة، خوارزمية، لغة الآلة، لغة برمجة إجرائية، لغة برمجة غرضية التوجه، تصميم من القمة إلى القاعدة، شبه التشفير.

ملخص:

يستعرض هذا القسم مفهوم البرنامج الحاسوبي من خلال عرضه لمراحل تطور العتاد الحاسوبي الصلب، وأنظمة تشغيله، وأساليب الترميز، ولغات البرمجة المستخدمة لتطوير الأنظمة البرمجية، بالإضافة إلى مراحل تطور منهجيات وأدوات تصميم وبناء هذه الأنظمة من خلال التركيز على شبه التشفير كأداة مساعدة على تصميم البرامج الصغيرة الحجم. كما يستعرض أنماط لغات البرمجة وأنواعها كلغات البرمجة الإجرائية ولغات البرمجة الغرضية التوجه.

أهداف تعليمية:

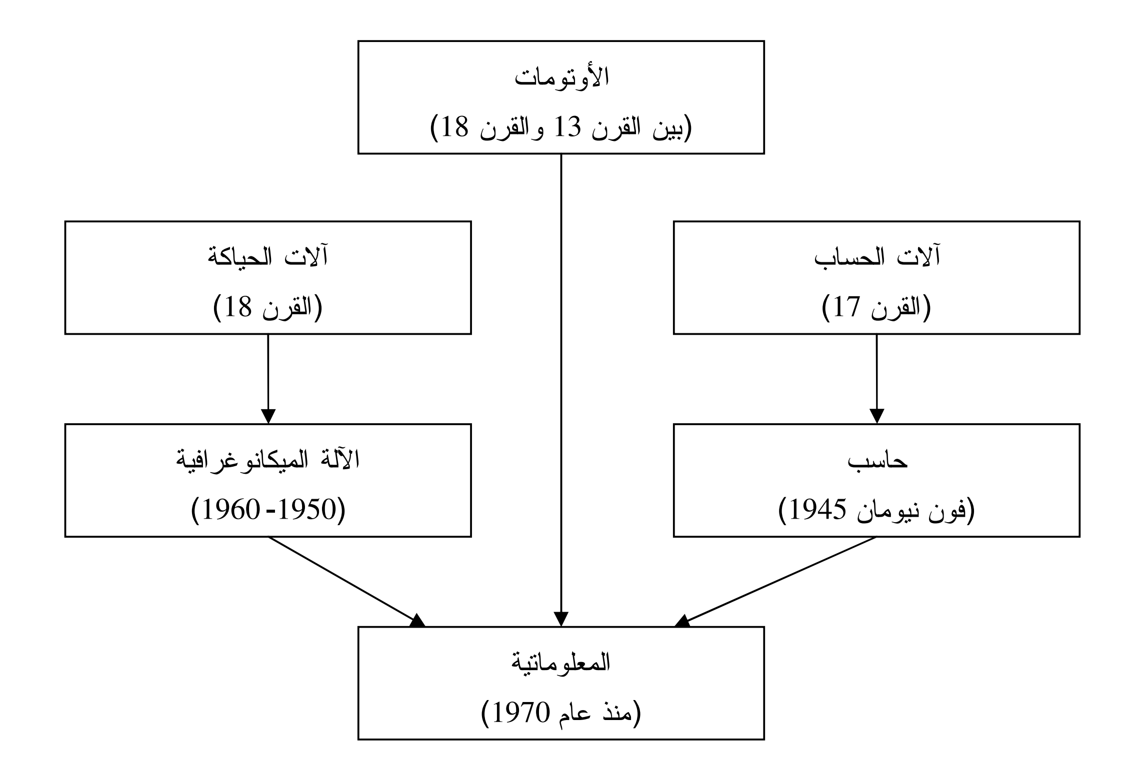
يتعرف الطالب في هذا الفصل على:

- نظام التشغيل، البرنامج الحاسوبي؛
- المترجم؛
- أنماط الترميز؛
- أنماط لغات البرمجة؛
- مفهوم الخوارزمية وأمثلة عنها؛
- منهجية تطوير البرمجيات؛
- شبه التشفير؛
- المخططات التدفقية.

الحاسوب – الآلة

يبدأ تاريخ المعلوماتية وعلوم الحاسوب مع اختراع الآلة التي ارتبط تطورها بثلاثة خطوط فكرية أساسية جرى التعبير عنها بثلاثة أنماط من الآلات:

- الآلة الحاسبة؛
- الأوتومات؛
- الآلة القابلة للبرمجة.



يبدأ تاريخ المعلوماتية وعلوم الحاسوب مع اختراع الآلة التي ارتبط تطورها بثلاثة خطوط فكرية أساسية مثلت ما ينتظره الإنسان من الآلة التي يخترعها ويطورها، وجرى التعبير عنها بثلاثة أنماط من الآلات: الآلة الحاسبة، الأوتومات، الآلة القابلة للبرمجة.

الآلة الحاسبة:

اخترع Pascal في القرن السابع عشر آلة حساب دعاها La Pascaline لتنفيذ عمليتي الجمع والطرح، وقد اعتمد في بنائها على المحسب الصيني القديم والذي يرجع تاريخه إلى مئات الأعوام قبل الميلاد. ومع نهاية القرن السابع عشر حسَّن Leibniz آلة باسكال بإضافة عمليتي الضرب والقسمة عليها.

الأوتومات:

بدأ تطوير الآلات الميكانيكية التي كانت تُستخدم في العمليات العسكرية وفي الساعات الفلكية منذ القرن الثاني عشر الميلادي واستمرت هذه الآلات الميكانيكية بالتطور حتى القرن الثامن عشر. وتظهر نماذج هذه الآلات وأساليب عملها في التصاميم التي تركها Leonardo De Vinci للكثير من الآلات العسكرية والمدنية.

الآلات القابلة للبرمجة:

بدأ مفهوم الآلات القابلة للبرمجة بالظهور مع اختراع آلات حياكة النسيج. وقد شهد هذا النوع من الآلات قفزة على يد Jaquard الذي عاش بين عامي 1752 و 1834 وصمم أول آلة حياكة قابلة للبرمجة (ميكانيكياً)، حيث استُعملت نفس التقنية بعدها لبناء العديد من الآلات الحربية.

وقد نتجت العلوم المعلوماتية عن اندماج الأفكار والمعارف التي جرى تحصيلها من تطوير الآلات الآنفة الذكر

تطور الحاسوب والديمقراطية المعرفية

من معلوماتية المشعوذين إلى معلوماتية الجميع

	الجيل الأول 1954-1945	الجيل الثاني 1965-1955	الجيل الثالث 1973-1966	الجيل الرابع منذ عام 1974
المكونات	صمامات مُفرّغة (Tubes)	ترانزيستورات (Transistors)	دارات مُدمَجة متكاملة (Integrated Circuits)	VLSI
الذواكر	Ferrite Core Memory	Ferrite Core memory	دارات مُدمَجة متكاملة (Integrated Circuits)	VLSI
زمن المعالجة	10^{-2} ثانية	10^{-3} ثانية	10^{-6} ثانية	10^{-9} ثانية
نظام التشغيل	بدائي	برنامج وحيد مع معالجة برنامج واحد	عدة برامج مع معالجة برنامج واحد	الاشتراك بالمعالج مع معالجة عدة برامج بآن واحد

مرّ تطور الحاسوب بعدة مراحل يجري عادةً تصنيفها تحت إسم أجيال الحاسوب، وتُقسَم هذه الأجيال إلى:

الجيل الأول (1945-1954):

استخدمت دارات هذه الحواسيب الصمامات المُفرَّغة، وكانت ضخمة بحيث يصعب تحريكها. كما كانت تعليمات نظام التشغيل تُخزَّن داخلياً، وكان لا بد من إضافة صمامات وأسلاك حديدية جديدة عند بروز الحاجة لإضافة تطبيقات جديدة.

قامت شركة **IBM** بتصنيع أول حاسوب ضخم وإسمه **IBM701**. كما شهد عام 1951 تصنيع أول حاسوب أميريكي تجاري، وهو **UNIVAC-1** والذي كان الهدف منه تجميع المعلومات السكانية الإحصائية. كان حاسوب **UNIVAC-1** يحتاج إلى طابق بناء ضخم، وكان وزنه ثمانية أطنان، ويحتوي على أكثر من ثمانية آلاف صمام مُفرَّغ.

في ذلك الوقت كان استخدام الحاسوب محصوراً في بعض المراكز العسكرية الكبرى في بعض الدول العظمى.

الجيل الثاني (1955–1965):

استخدمت هذه الحواسيب الترانزيستورات في تنفيذ العمليات الحسابية، واحتوت على ذاكرة مغناطيسية، واستخدمت أقراصاً وأشرطة مجدولة ممغنطة لتخزين المعطيات. وقد سمح هذا التصميم بتخزين البرامج، وأصبح بإمكان مدير النظام إدخال تعليمات التنفيذ، اعتماداً على لوحة مفاتيح. وقد ظهرت في هذه الفترة أولى لغات البرمجة كالــ Fortran التي كانت تُستخدم في تنفيذ الأعمال الحسابية، والــ Cobol والتي كانت تُستَخدَم في أتمتة بعض الأعمال الإدارية والمكتبية. حينها، كانت الدول الكبيرة والغنية فقط قادرة على اقتناء واستخدام الأدوات الحاسوبية.

الجيل الثالث (1966–1973):

استخدمت هذه الحواسيب الدارات المُدمَجة والمُتكاملة. لكن الحواسيب لم تكن متوافقة فيما بينها، بحيث كانت الطرفيات مصممة للاستخدام على حاسوب وحيد ولا تعمل مع أي حاسوب آخر، وكان يتوجب إعادة كتابة وترجمة نظم التشغيل الخاصة بأحد الحواسيب لكي تعمل على حاسوب آخر.

أنتجت شركة **DEC (Digital Equipment Corporation)** الأميريكية حاسوب **PDP-8**. كان هذا أول حاسوب بحجم صغير نسبياً، وكان هدفه التحكم في عمليات المعالجة الصناعية والعلمية، إلا أن التطبيقات الأخرى ذات الأغراض المختلفة بدأت بعد ذلك بالتوافر في الأسواق تدريجياً.

كما طورت شركة **AT&T** الأميريكية بالتعاون مع مختبرات **Bell** نظام التشغيل **UNIX**، والذي يعتمد على تعدد المستخدمين.

منذ ذلك الوقت، صار الحاسوب في متناول عدد أكبر من الدول والبلدان وصار بالإمكان اقتناؤه من قبل الجامعات والمؤسسات الحكومية والخاصة الكبيرة لاستخدامه في الأعمال العلمية.

الجيل الرابع (منذ عام 1974):

تميزت هذه الحواسيب بالدارات المتكاملة المُدمَجة ذات الأحجام الصغيرة جداً، وكانت سرعاتها عالية، وكانت الأجهزة التي تحويها موثوقة، ولها شاشات مرئية، ومساحات تخزين واسعة.

حازت شركة مايكروسوفت على رخصة لاستخدام نظام UNIX، وبدأت بتطوير نسخة من نظام Xenix للحواسيب الشخصية.

واعتمدت شركة IBM عام 1980 على مهندسين هما Paul Allen و Bill Gates لابتكار نظام تشغيل حاسوب شخصي جديد حيث قاموا بشراء حقوق نظام تشغيل بسيط استخدموه كنموذج لنظام تشغيل مبدئي يدعى DOS. وقد سمحت IBM لكل من Paul Allen و Bill Gates بالإحتفاظ بحقوق تسويق نظام التشغيل MS-DOS، إضافة إلى حق استخدام الإسم التجاري DOS. كان نظام MS-DOS أو Microsoft Disk Operating System في البداية نظام تشغيل بسيط، مصمماً لتشغيل برنامج واحد، في آن واحد، ولمستخدم وحيد.

في عام 1984، سوقت شركة Apple حاسوب Macintosh على نطاق واسع. وقد استخدمت حواسيب Apple Macintosh واجهات بيانية رسومية تعمل بالمؤشر، بدلاً من لوحة المفاتيح، كما كان الأمر عليه في نظام DOS.

ي نفس الوقت أصدرت Microsoft النسخة الأولى من نظام Windows وطورته عبر عقدين لتحوله من نظام خاص بحاسوب شخصي إلى نظام يمكن استخدامه ضمن شبكات حاسوبية في المؤسسات.

وفي عام 1991 طور Linus Torvald نظام التشغيل LINUX المجاني ذو الرماز المفتوح الذي يعمل على الحواسيب الشخصية والمشابه لنظام UNIX من حيث المكونات، بهدف محاربة احتكار Microsoft لأنظمة الحواسيب الشخصية.

في عصرنا هذا، أصبح الحاسوب أداة متوفرة للجميع ولم يعد مقتصراً على مجموعة من الأخصائيين المشعوذين !!!

نظم التشغيل

- **تعريف:**

يُعرَّف نظام التشغيل بأنه برنامج يدير عتاد الحاسوب بحيث يوفر البرمجيات والتطبيقات الضرورية لتشغيل هذه العتاديات، كما يعمل كوسيط بين المستخدم والحاسوب بحيث يسمح للمستخدم باستثمار الحاسوب وتطبيقاته.

- **أسلوب تصميم نظام التشغيل:**

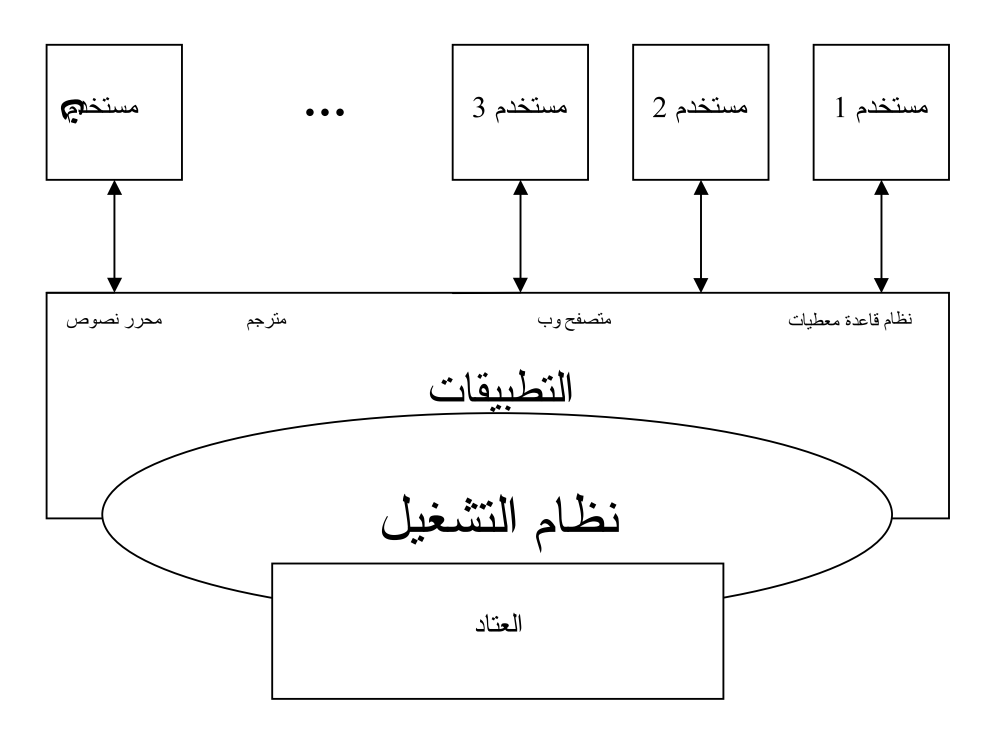
يختلف تصميم نظام التشغيل حسب البيئة التي يُفترض أن يعمل عليها، إذ يُصمم نظام التشغيل الذي يعمل على المخدمات على نحو يستطيع فيه استثمار العتاديات بالشكل الأمثل، في حين يصمم نظام التشغيل المُعدّ للعمل على الحاسبات الشخصية ليدعم تطبيقات متنوعة. بالتالي نلاحظ اختلاف وجهة التصميم لتكون إما ملائمة للمُستخدم النهائي في حالة الحواسب الشخصية، أو فعالة في استثمارها للعتاديات في حالة المخدمات.

الحاسوب ونظام التشغيل

مكونات النظام الحاسوبي:

- العتاديات؛
- نظام التشغيل؛
- التطبيقات؛
- المستخدمين.

يمثل الشكل التالي بنية توضيحية للنظام الحاسوبي، ويُبين توضع نظام التشغيل ضمن تلك البنية:



مهمة نظام التشغيل:

- نظام التشغيل كمحصص للموارد؛
- نظام التشغيل كبرنامج تحكم؛

نظام التشغيل كنواة. يعتبر نظام التشغيل جزءاً هاماً من كافة الأنظمة الحاسوبية، بحيث يمكن أن نقسم النظام الحاسوبي إلى أربعة مكونات رئيسية وهي:

- العتاديات؛
- نظام التشغيل؛
- التطبيقات؛
- المستخدمين.

يتولى نظام التشغيل مهمة الإشراف والمراقبة وتوفير البيئة الملائمة للتطبيقات والمستخدمين لكي يُنفذوا أعمالهم ويستثمروا موارد الحاسوب وتطبيقاته. إذ تشكل العتاديات في النظام الحاسوبي الموارد التي يجري الاعتماد عليها عند استثمار الحاسوب، وهي تشمل وحدة المعالجة المركزية، والذاكرة، وتجهيزات الدخل/خرج وغيرها، في حين تُعبِّر التطبيقات عن الأدوات التي يستخدمها المستثمرون لاستثمار الموارد.

يمكن النظر إلى نظام التشغيل كمحصص للموارد، وكنظام تحكُم، و كنواة لتشغيل التطبيقات الحاسوبية:

- نظام التشغيل كمحصص للموارد:

  يتكون النظام الحاسوبي من العديد من الموارد العتادية والبرمجية (وحدة معالجة مركزية، وحدات خزن معطيات، ذاكرة رئيسية ...الخ)، حيث يتولى نظام التشغيل مهمة إدارة تلك الموارد وتوزيعها على المستخدمين بشكل مُنصِف يضمن فعالية أداء النظام الحاسوبي. وتبرز أهمية وقدرة نظام التشغيل على الإدارة في أسلوب معالجته للطلبات التي يمكن أن تؤدي إلى تعارض في استخدام الموارد.

- نظام التشغيل كبرنامج تحكم:

  يمكن النظر إلى نظام التشغيل كبرنامج يتحكم بكيفية تنفيذ برامج المستخدمين بهدف منع حدوث الأخطاء، ومنع الاستخدام غير السليم للحاسب وخاصة فيما يتعلق باستخدام تجهيزات الدخل/خرج والتحكم فيها.

- نظام التشغيل كنواة:

  إن المفهوم الذي يعتبر نظام التشغيل أداة تحصيص أو أداة تحكم يُولِّد بالضرورة تصوراً حول مكونات نظام التشغيل من البرمجيات، لذا يجدر بنا التنويه إلى التعريف الأكثر شيوعاً لنظام التشغيل –الذي يُطلق عليه اسم النواة– والذي يشير لنظام التشغيل **<u>على أنه البرنامج الذي يكون بحالة تنفيذ دائمة والذي تعمل تحت إشرافه التطبيقات البرمجية الأخرى.</u>**

التصنيفات الرئيسة لأنواع نظم التشغيل وتطورها

- نظم المهمة الوحيدة؛
- نظم المهمات المتعددة ونظم المشاركة بزمن المعالج؛
- نظم الحواسيب الشخصية؛

النظم الموزعة؛تطورت نظم إدارة الحواسيب تطوراً كبيرا منذ أن نشأت وحتى الآن، سواء كان ذلك التطور يؤثر على طبيعة نظام التشغيل بحد ذاته، أو كان يعبر عن جيل آخر من الأنظمة يقدم خدمات مغايرة أكثر تطوراً وتنوعاً من حيث دعمها للتطبيقات المختلفة وما تقدمه من مهمات، تجارية كانت أم علمية؛

لقد مرت دورة حياة نظم التشغيل بالعديد من المراحل فبدأت من خلال النظم ذات المهمة الوحيدة، وتطورت بعد ذلك لتصبح نظماً تدعم عدة مهمات في آن واحد، ثم بدأت تتشارك بالموارد كالمعالج أو الذاكرة، وترافق ذلك مع تطور أجيال الحواسيب الشخصية التي انتشرت انتشاراً واسعاً بين المستخدمين؛

تُعبِّر نظم المهمة الوحيدة عن نظم التشغيل البسيطة التي كان الحاسوب فيها يقوم بتنفيذ تطبيق واحد فقط، وتُمثِّل هذه النظم الشكل الأول لنظم التشغيل عند بداية ظهورها، حيث كانت الحاسبات في ذلك الوقت ذات حجوم ضخمة جداً وكانت تُدار من خلال واجهات تعليمات خاصة، أما أدوات الدخل / خرج فقد كانت تتمثل بقارئات البطاقات المثقبة وسواقات الأشرطة، كما كانت وسائط التخزين تتمثل عموماً بالبطاقات المثقبة؛

وتُعبِّر نظم المهمات المتعددة عن نظم التشغيل التي تستثمر الموارد على نحو يزيد من معدل استخدام وحدة المعالجة المركزية وبحيث يتم تنفيذ إجرائية في كل لحظة؛ يجري تخزين الأعمال في قرص تخزين، كما يجري انتقاء مجموعة من تلك الأعمال ونقلها إلى الذاكرة لكي يجري تنفيذها معاً، ولا يجري نقل كافة الأعمال المخزنة لأنه غالباً ما تكون المعطيات المخزنة على القرص أكبر من سعة التخزين في الذاكرة؛ تسمى عملية انتقاء الأعمال التي ينبغي اختيارها أولاً بجدولة الأعمال.

ومع الانخفاض الكبير في تكلفة المعالجات أصبح بالإمكان امتلاك المستخدم لنظامه الحاسوبي الخاص به. أُطلق على هذا النوع من النظم اسم نظم الحواسيب الشخصية؛ وتزامن ظهور هذا النوع من النظم مع تطور التجهيزات الحاسوبية تطوراً كبيراً على صعيد الشكل والأداء، فعلى سبيل المثال تغيرت معظم أساليب الدخل التي كانت سائدة لتتحول إلى طرائق استخدام للوحة المفاتيح والفأرة، كما تغيرت معظم أساليب الخرج لتصبح من خلال شاشات عرض أو طابعات صغيرة الحجم عالية الأداء؛

- يعتمد الاتجاه الحالي في تصميم نظم الحواسيب على مفهوم توزيع الحسابات بين عدة معالجات، يختلف هنا المفهوم المطروح عن مفهوم النظم التفرعية من مبدأ أن المعالجات لا تشترك بالذاكرة أو بالميقاتية إذ يمتلك كل معالج منها ذاكرته المحلية الخاصة، كما يتم التخاطب بين المعالجات من خلال أسلوب اتصال مناسب كشبكة محلية أو خطوط هاتف أو أية وسيلة أخرى. يُطلق على هذا النوع من النظم اسم النظم الموزعة. يمكن أن تختلف المعالجات المكونة للنظام الموزع حجماً أو أداءً، فيمكن أن تكون عبارة عن معالجات أو محطات عمل أو حواسب شخصية أو حتى منصّات، كما يمكن الإشارة إليها بأسماء مختلفة كمواقع وب أو كعقد شبكية، حيث تختلف التسمية بحسب السياق الذي يتم فيه الإشارة إلى تلك المعالجات

ترميز المعلومات

نُعرِّف الترميز على أنه تابع تقابل بين معلومة وبين سلسلة من 0 و 1 تمثل هذه المعلومة وتكون قابلة للتخزين ضمن الآلة.

أنواع الترميز:

- ترميز المحارف: الترميز ASCII (American Standard Code for Information Interchange)، و الترميز UNICODE (Universal Code)
- ترميز الأعداد الصحيحة: الترقيم

تعريف:

تكون المعلومات المُخزَّنة في الحاسوب على شكل سلسلة من 0 و 1 (اللغة الثنائية)، وبما أن الإنسان لا يتكلم اللغة الثنائية، نحتاج لترجمة تعليمات المستثمر المكتوبة بلغة برمجية خاصة، إلى هذه اللغة الثنائية. لذا نُعرِّف الترميز على أنه تابع تقابل بين معلومة، وبين سلسلة من 0 و 1 تمثل هذه المعلومة وتكون قابلة للتخزين ضمن الآلة.

ترميز المحارف: الترميز ASCII (American Standard Code for Information Interchange)، و الترميز UNICODE (Universal Code)

يعتبر الترميز ASCII أحد أهم أساليب الترميز المُتَّبعة في الأنظمة الحاسوبية. يسمح الترميز ASCII بشكله المُعدّل بترميز أي محرف على 8 بت. لذا يمكننا اعتماداً على مثل هذا الترميز، تمثيل 2^8 محرف (أي 256 محرف) مما يسمح بتمثيل الأبجديات الأوروبية كالإنكليزية، والفرنسية، والإسبانية،... الخ، بالإضافة إلى المحارف الخاصة كالأرقام وأحرف التنقيط وغيرها.
جرى إدخال تعديلات حديثة على أنظمة الترميز ضمن الأنظمة الحاسوبية بحيث سمحت بتمثيل المحارف على 16 بت، ودُعيت بالترميز العمومي (Universal Code)، مما ساعد على توفير إمكانية تمثيل 65536 حرف، وأدى لفتح المجال أمام تمثيل الأحرف العربية، والصينية، والكورية وغيرها.

ترميز الأعداد الصحيحة: الترقيم

يمكن ترميز الأعداد الصحيحة كمحارف، إلا أن مثل هذا الترميز سيُعقِّد تنفيذ العمليات الحسابية على هذه الأعداد ضمن الأنظمة الحاسوبية. بالنتيجة، يمكن للحاسوب التعامل مع القيم الرقمية على نحوٍ أسهل إذا جرى وضع ترميز خاص لها. ندعو هذا الترميز بالترقيم.
عادةً، يجري التعامل مع القيم الرقمية الصحيحة كقيم عِشرية: فالرقم 5، والرقم 8، والرقم 90 هي أرقام صحيحة ممثلة على قاعدة الترقيم العِشّري بحيث تكون الأرقام محصورة بين 0 و 9 وتكون قيم الأعداد محسوبة وفق القاعدة العشرية. فعندما نكتب العدد 5769 وفق القاعدة العِشّرية، يشير ترتيب الأرقام إلى قوة الرقم 10 المرتبطة بالرقم وهي في حالتنا:

$5 \rightarrow 10^3$
$7 \rightarrow 10^2$
$6 \rightarrow 10^1$
$9 \rightarrow 10^0$

وتكون قاعدة احتساب القيمة العشرية الموافقة لهذه الارتباطات من الأعلى إلى الأسفل:

$(5 * 10^3) + (7 * 10^2) + (6 * 2^1) + (9 * 10^0) = 5769$

يمكننا تعميم هذه القاعدة على أي قاعدة b مهما تكن b سواء كانت b=2 أو b=10 أو b=16. فعندما تكون b=10 ندعو قاعدة الترقيم، قاعدة عِشرية، وتكون الأرقام التي تؤلف الأعداد محصورة بين 0 و 9، وعندما تكون b=2 ندعو قاعدة الترقيم، قاعدة ثُنائيّة، وتكون الأرقام التي تؤلف الأعداد محصورة بين 0 و 1، وعندما تكون b=8 ندعو قاعدة الترقيم قاعدة ثُمانّية وتكون الأرقام التي تؤلف الأعداد محصورة بين 0 و 7.

بالنتيجة، تكون ارتباطات الأرقام التي تؤلف العدد 010010 الممثل ثنائياً كما يلي من اليسار إلى اليمين:

$2^5 \rightarrow 0$
$2^4 \rightarrow 1$
$2^3 \rightarrow 0$
$2^2 \rightarrow 0$
$2^1 \rightarrow 1$
$2^0 \rightarrow 0$

وتكون قاعدة احتساب القيمة العشرية الموافقة لهذه الارتباطات من الأعلى إلى الأسفل:

$(0 * 2^5) + (1 * 2^4) + (0 * 2^3) + (0 * 2^2) + (1 * 2^1) + (0 * 2^0)= 18$

البرامج الحاسوبية

تجري كتابة البرامج الحاسوبية على شكل تعليمات وتراكيب حسابية ومنطقية، وذلك باستخدام إحدى لغات البرمجة، مثل #C. وتجري ترجمة هذه التعليمات والتراكيب إلى سلاسل من الرموز الرقمية الثنائية التي تعبر عن شيفرة يفهمها الحاسوب وتدعى لغة الآلة.

تتضمن عملية البرمجة كتابة مجموعة من التعليمات على نحو متسلسل، بحيث يجري الحصول على النتيجة المطلوبة عند تنفيذ التعليمات المتسلسلة في الحاسوب. ويجري تخزين البرامج على القرص الصلب وتحميلها في الذاكرة الحية عند بدء تنفيذ البرنامج وذلك لتسريع عملية التنفيذ.

تكون وحدة المعالجة المركزية مسؤولة عن معالجة الشيفرة الناتجة عن عملية ترجمة البرنامج والمكتوبة بلغة الآلة من خلال:

- نقل المعطيات ضمن وحدة المعالجة، أو من وحدة المعالجة إلى الذاكرة، أو من الذاكرة إلى وحدة المعالجة، أو من وحدة المعالجة إلى الطرفيات؛
- تنفيذ العمليات الحسابية؛
- تنفيذ العمليات المنطقية؛

لغات البرمجة

هناك نوعان من اللغات البرمجية المستخدمة في الحواسب: اللغات منخفضة المستوى و اللغات عالية المستوى.

ترتبط اللغات البرمجية منخفضة المستوى بالعتاد الصلب (نمط وحدة المعالجة، نمط النواقل وسعتها، ... الخ) وتدعى عادةً بلغة المُجمِّع

وتَستخدِم رموزاً تمثل عمليات الحاسوب، ويتوجب ترجمة كافة الرموز المكتوبة بلغة المُجمِّع إلى لغة الآلة الممثلة بشيفرة وسلاسل ثنائية (مؤلفة من 0 و1). تجري عملية الترجمة باستخدام برامج خاصة تُدعى المُجَمِعات.

أما اللغات البرمجية عالية المستوى فتكون مستقلة عن العتاد الصلب، بحيث تجري كتابة البرامج بتعليمات وعبارات مشابهة للغة الإنكليزية. ولهذه اللغات عدة أصناف: اللغات الإجرائية، واللغات الوظيفية، واللغات غرضية التوجه، ولغات التوصيف. ويتوجب ترجمة كافة الرموز المكتوبة بلغة برمجة عالية المستوى إلى لغة الآلة الممثلة بشيفرة وسلاسل ثنائية (مؤلفة من 0 و1). تجري عملية الترجمة باستخدام برامج خاصة تُدعى المُترجمات.

اللغات البرمجية عالية المستوى: لمحة تاريخية

تضمن اللغات البرمجية عالية المستوى تحقيق مجالٍ واسعٍ من المهام البرمجية المختلفة.

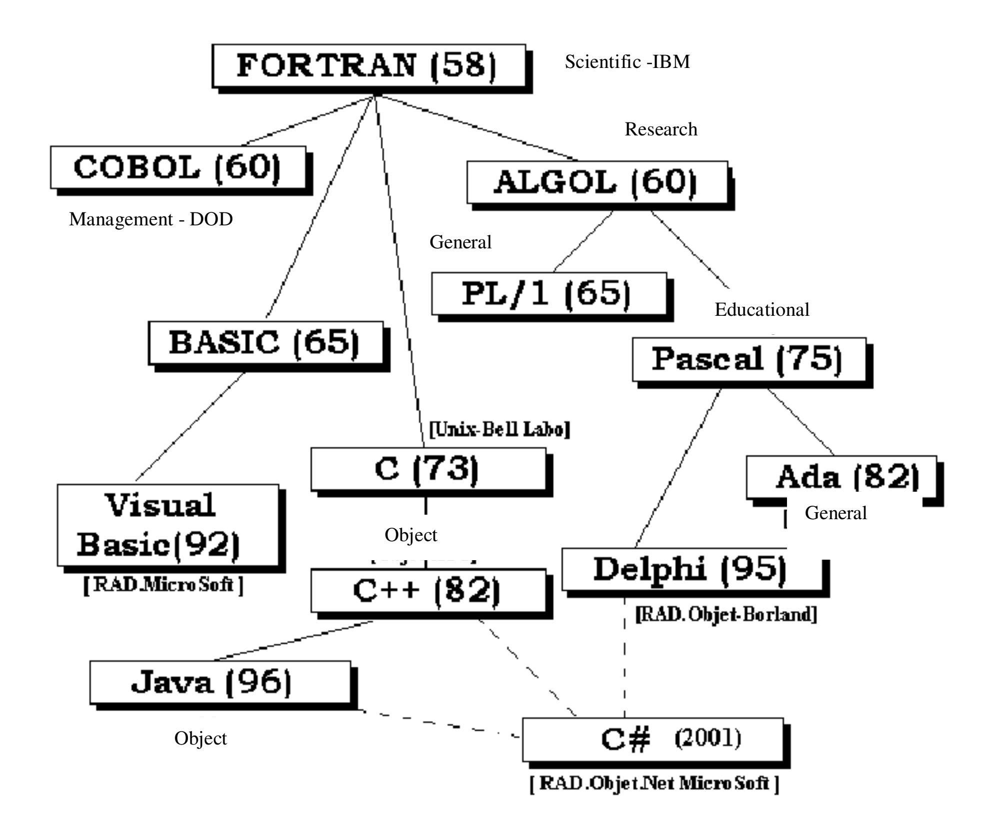
لقد جرى تطوير العديد من لغات البرمجة المختلفة على مر السنين بهدف تلبية الاحتياجات المتغيرة في تقنيات المعلومات:

- عام 1958: لغة Fortran؛
- عام 1964: لغة BASIC؛
- عام 1970: لغة ADA؛
- عام 1971: لغة Pascal؛
- عام 1972: لغة C؛
- عام 1986: لغة C++؛
- عام 1991: لغة Visual Basic؛
- عام 1995: لغة Java.

تضمن اللغات البرمجية عالية المستوى تحقيق مجالٍ واسعٍ من المهام البرمجية المختلفة. فبالرغم من أن معظم اللغات البرمجية عالية المستوى قد صممت خصيصا لمجالات تطبيقية عامة، كلغة Fortran التي صُممت كلغة عامة، إلا أنها استُخدِمت في تطبيقات محددة كلغة Fortran التي استُخدِمت في حل المشاكل الرياضية والحسابية.

لقد جرى تطوير العديد من لغات البرمجة المختلفة على مر السنين بهدف تلبية الاحتياجات المتغيرة في تقنيات المعلومات. ففي عام 1964 قام كل من John Kemeny وThomas Kurtz من جامعة Dartmouth بتطوير لغة BASIC الموجهة لكافة الأغراض. وفي عام 1970 طورت وزارة الدفاع الأميريكية لغة ADA وهي لغة خاصة ببرمجة الحواسب، وتضمنت هذه اللغة إمكانيات خاصة بتصميم أنظمة دفاعية لتوجيه القذائف العسكرية. وفي عام 1971 ابتكر Niklaus Wirth لغة البرمجة Pascal، وابتكر Dennis Ritchie عام 1972 لغة البرمجة C في مختبرات شركة Bell الأميركية. وجرى تطوير لغة C++ اعتماداً على لغة C في مختبرات Bell التابعة لشركة AT&T الأميريكية عام 1986 وباتت تعتبر واحدة من أكثر اللغات البرمجية ذات التوجه الغرضي استخداماً. وطورت Microsoft عام 1991 لغة Visual Basic التي تعد قوية في تطوير واجهات برمجية تعمل في بيئة نظم Windows. وتبعتها في عام 1995 شركة Sun Microsystems الأميريكية بتطويرها للغة Java التي تدعم برمجيات الانترنيت، بما في ذلك مايدعى Java Applets.

اللغات البرمجية عالية المستوى: اللغات الإجرائية (1)



مثال عن برنامج بلغة إجرائية هي لغة FORTRAN

```
C   *** FORTRAN ***
    integer i,j,k
    write(5,50)
5   format(2X,6HGood Day)
    i=15
    if (i.GT.10) goto 10
    read(6,80) j
6   format(i4)
    do 10 k=1,i-1,2
      j=j+1
10  continue

    end
```

اللغات البرمجية عالية المستوى: اللغات الإجرائية (2)

تتلخص المكونات الأساسية لمعظم اللغات الإجرائية بما يلي:

- التعليمات؛
- أنماط المعطيات والمعرّفات؛
- العمليات؛
- الدخل والخرج؛
- التوابع والإجرائيات.

تتلخص المكونات الأساسية لمعظم اللغات الإجرائية بما يلي:

التعليمات:

للغات البرمجية معجم محدد من الكلمات والتعليمات الخاصة، ومثال ذلك: تعليمة الإنتقال GOTO، والإسناد LET والإنهاء END، بالإضافة إلى التعليمات الشرطية IF، والحلقية WHILE.

أنماط المعطيات:

تعبر أنماط المعطيات عن حجم الذاكرة المخصصة لتخزين قيمة محددة أو مجموعة من القيم، إذ يعبر نمط العدد الصحيح أو Integer على سبيل المثال، عن 16 بت أو 32 بت من مساحة الذاكرة المخصصة لتخزين عدد صحيح. ندعو الأنماط الأساسية (أعداد صحيحة، أعداد حقيقية، محارف، ... الخ) بالأنماط البسيطة، في حين ندعو مصفوفات الأعداد وسلاسل المحارف بالأنماط المُركبة.

العمليات:

توجد، بالإضافة إلى التعليمات، رموز أخرى تدعى بالعمليات يتم استخدامها للإشارة إما إلى عملية حسابية، أو إلى علاقة منطقية.

الدخل والخرج:

يتم تنفيذ عملية الدخل باستخدام تعليمات محددة مثل READ، كما يجري تنفيذ عملية الخرج باستخدام تعليمات محددة مثل WRITE أو PRINT.

يكون الوسيط الافتراضي المُستَخدَم في إدخال المعطيات هو لوحة المفاتيح، إلا إذا قام المُبرمج بتعريف وسيط آخر. أما الوسيط الافتراضي المُستَخدَم في إخراج المعطيات، فغالباً ما يكون شاشة الحاسوب ما لم يقم المبرمج بتعريف وسيط آخر.

الإجرائيات والمكتبات:

تتكون الإجرائية من سلسلة من التعليمات التي تُعدّ جزءاً من البرنامج، لكنها تكون مستقلة عن السلسلة الرئيسية لتعليمات البرنامج التي يجري تنفيذها. لا تشكل الإجرائية بحد ذاتها برنامجاً مستقلاً، ويجري استدعاؤها بوساطة البرنامج الرئيسي حين الحاجة لها فقط. تمتلك كل لغة برمجة مجموعة من الإجرائيات المُعرَّفة مُسبقاً ضمن لوائح جاهزة ندعوها مكتبات برمجية.

اللغات البرمجية عالية المستوى: اللغات الوظيفية

تعتمد العمليات في اللغات الوظيفية على توابع رياضية ومنطقية وعلى وجود قاموس من التوابع المُعرَّفة مُسبقاً وعلى آلية لبناء توابع جديدة من قبل المُبرمج.

إذ تقوم لغة LISP (List Processing) مثلاً وهي إحدى اللغات الوظيفية، بالتعامل مع كافة عناصر البرنامج على أنها جزء من سلسلة وتوفر التوابع اللازمة لمعالجة هذه السلاسل ومسحها.

فعلى سبيل المثال يجري التعبير عن عملية على عددين صحيحين بالشكل (op 2 3) حيث يجري التعامل مع التعبير السابق على أنه سلسلة من 3 محارف، ويجري تنميط الرقمين 2 و 3 واعتبارهما عددين صحيحين مباشرةً، ويجري التعامل مع أسم التابع op على أنه رمز خاص يمكن تعريف نتيجة تطبيقه على عددين صحيحين في مكان آخر.

كما يمكن التعبير عن عملية مُعرّفة مُسبقاً مثل عملية الجمع على عددين صحيحين مثل 2 و 3 بالشكل (+ 2 3).

اللغات البرمجية عالية المستوى: اللغات المنطقية

جرى اشتقاق اللغات المنطقية وعلى رأسها لغة PROLOG (PROgrammation en LOGique) من مفاهيم الذكاء الصنعي وتقنياته.

تجبر اللغة المنطقية المُبرمج على التفكير بأسلوب المنطق الذي يعتمد على الانطلاق من مجموعة من المقدمات للوصول إلى مجموعة من النتائج التي يمكن أن تُصبح بدورها مقدمات تُغني المُقدمات المُعرَّفة مُسبقاً.

اعتماداً على هذا المبدأ يمكن باستخدام لغة منطقية تعريف المُقدمتين: (كل إنسان فان)، و (سُقراط إنسان)، ويمكن اعتماداً على محرك خاص بالتنفيذ، استخلاص النتيجة وهي (سقراط فان)، بحيث يمكن إضافة النتيجة إلى المقدمات لإغنائها.

ندعو المحرك الذي يستخلص النتائج من المقدمات **بمحرك استدلال**.

اللغات البرمجية عالية المستوى: اللغات الغرضية التوجه

تعتمد البرمجة الغرضية التوجه على أساس بناء النظام البرمجي على شكل مجموعة من الأغراض التي تتواصل فيما بينها من خلال رسائل اعتماداً على توابع وإجرائيات مرتبطة بالأغراض ندعوها **الطرائق**.

يكافئ مفهوم **الغرض** في التصميم الغرضي التوجه مفهوم المتحول في اللغة الإجرائية العادية، في حين يلعب مفهوم **الصف** في اللغة الغرضية التوجه، دور النمط في اللغة الإجرائية.

تُعتبر لغات مثل ++C، Java، #C من أشهر اللغات الغرضية التوجه، وسنستعرض مفاهيم التصميم والبرمجة الغرضية التوجه لاحقاً في

فصل خاص كما سنركز في فصولنا اللاحقة على لغة #C.

المترجمات

يدعى البرنامج الذي يقوم المبرمج بكتابته بإحدى اللغات البرمجية، بإسم **البرنامج المصدري** أو البرنامج الأصلي. ولكي يتمكن الحاسوب من تنفيذ البرنامج، ينبغي على المبرمج أن يقوم بترجمة البرنامج إلى لغة الآلة وبناء **برنامج تنفيذي** مكافئ للبرنامج المصدري.

تجري عملية الترجمة بوساطة **مُترجِم** خاص بلغة البرمجة المُستخدمة لكتابة البرنامج وخاص بنظام التشغيل الذي يعمل عليه المُبرمج، حيث يقوم المترجم بتحويل البرنامج الأصلي إلى برنامج تنفيذي.

يجري الإعلان عن الأخطاء التي يرتكبها المُبرمج عند كتابته لبرنامجه أثناء الترجمة. كما ينبغي على المترجم أن يتمكن من الدخول إلى مكتبة الإجرائيات الجاهزة التي تتضمن العديد من البرامج والإجرائيات اللازمة لتنفيذ العمليات الحسابية، وعمليات الدخل والخرج، وغيرها. وحيثما أشار البرنامج المصدري لإحدى هذه الإجرائيات، أو احتاج لتنفيذ عملية محددة، يقوم المترجم بالتأكد من إضافة الإجرائية المكتوبة بلغة الآلة إلى البرنامج التنفيذي.

أسئلة

أولاً- حدد اللغات الإجرائية من بين اللغات التالية:

- C++
- Pascal
- COBOL
- Java
- C#

ثانياً- حدد اللغات الغرضية التوجه من بين اللغات التالية:

- C++
- Pascal
- PROLOG
- Java
- LISP

ثالثاً- حدد العنصر الغريب من بين العناصر التالية:

- متحول
- غرض
- نمط
- إجرائية
- مكتبة إجرائيات

رابعاً- عرّف المُترجم، وحاول باستخدام الإنترنت البحث عن تعريف لمعنى كلمة مُفسِّر (Interpreter)، وحدد الفرق بين المُترجم

والمُفسر وأعط مثالاً عن بعض المُفسرات.

خامساً– حدد التسلسل الزمني التاريخي لظهور اللغات التالية من الأقدم إلى الأحدث:

- PASCAL
- FORTRAN
- C
- JAVA
- COBOL

الخوارزميات

تُعتبر الخوارزميات أدوات رياضية مساعدة على إيجاد حلول منهجية للعديد من المسائل. ويجري تعريف الخوارزمية على أنها سلسلة من التعليمات، أو الخطوات الإجرائية لحل مشكلة ما. إذ يمكن توصيف أي مشكلة، حتى تلك التي لاتتعلق بالحوسبة، باستخدام أحد أساليب وضع الخوارزميات.

مثال 1: خوارزمية استخدام الحاسب:

1. اضغط على زر التشغيل؛
2. انتظر ظهور شاشة الاستقبال؛
3. إذا كان من الضروري أن تُعرِّف عن نفسك: أدخل إسم حسابك وكلمة مرورك؛
4. ابحث عن أيقونة البرنامج الذي تريد تشغيله وانقر عليها نقرتين بالفأرة.

مثال 2: خوارزمية تحضير بيضة مقلية:

1. ضع الوعاء على النار؛
2. أضف مقدار نصف ملعقة صغيرة من الزبدة؛
3. انتظر ذوبان الزبدة؛
4. اكسر البيضة وضع محتواها ضمن الوعاء؛
5. انتظر حتى تنضج البيضة.

مثال 3: خوارزمية تحديد الرقم الأكبر من مجموعة أرقام:

1. احتفظ بالأرقام ضمن جدول؛
2. أدخل أول أرقام الجدول؛
3. أدخل ثاني أرقام الجدول؛
4. إذا كان الرقم الأول أكبر من الرقم الثاني: احتفظ بالأول ضمن الجدول واحذف الثاني؛
5. وإلا: احتفظ بالثاني ضمن الجدول واحذف الأول؛
6. العودة إلى الخطوة 1 وتكرارها حتى يتبقى رقم واحد في الجدول؛
7. الرقم المتبقي في الجدول هو الرقم الأكبر.

مثال 4: خوارزمية عدّ الأرقام الموجودة ضمن خانات جدول:

بفرض أن لدينا جدول يحوي على عدد من الخانات. ولنفرض أنه قد جرى ملئ عدد من الخانات بأرقام في حين بقيت بقية الخانات فارغة. سنضع خوارزمية لعدّ الخانات المشغولة.

1. احتفظ ضمن عداد بالقيمة 0؛
2. ابدأ بالخانة الأولى؛
3. تحقق من الخانة؛
4. إذا كانت الخانة مملوءة إجمع القيمة واحد إلى القيمة المحتواة ضمن العداد؛
5. إذا انتهت الخانات اذهب إلى الخطوة 8؛
6. إذا لم تنته الخانات انتقل لفحص الخانة التالية؛
7. العودة إلى الخطوة 3؛
8. أخرج رقم العداد الذي يعبِّر عن عدد الخانات المشغولة.

نشاطات للمناقشة والحلّ خلال الجلسات

أولاً- حساب النسبة المئوية:

ضع خوارزمية تسمح لك بإدخال عددين واحتساب ما يمثله العدد الأصغر كنسبة مئوية من العدد الأكبر.

مثال: في حال إدخال العددين 5 و3، يمثل 3 النسبة 60% من 5.

ثانياً- ترتيب جدول:

بفرض أن لديك جدولاً يحتوي على 5 أعداد. ضع خوارزمية تساعدك على ترتيب الجدول بحيث تظهر أعداده من الأكبر إلى الأصغر.

مثال:

في حال كان الجدول كما يلي:

9
90
4
80
10

فيجب أن نحصل بتطبيق الخوارزمية على:

90
80
10
9
4

ثالثاً– حساب المتوسط الحسابي لعدة أرقام:

اعط خوارزمية حساب المتوسط الحسابي لعدة أعداد (المتوسط الحسابي لعدة أعداد هو حاصل تقسيم مجموع قيم الأعداد على عددها).

مثال: في حال أدخلنا الأعداد 6، 7، 20، 13 نحصل على متوسط حسابي هو 9 كمايلي:

Average=(6+7+20+13)/4=36/4=9

رابعاً – خوارزمية عدّ:

بفرض أن لديك جدولاً يحتوي على 10 خانات مملوءة بأعداد، أعط خوارزمية عدّ الأعداد المُخزنة في الجدول والتي تكون قيمتها أصغر من 10.

مثال: لدينا في الجدول التالي 4 أعداد أصغر من 10

6	80	90	1001	9	0	300	400	50	4

خامساً – المُضاعَف المُشترَك البسيط:

أعط خوارزمية حساب المُضاعَف المُشتَرَك البسيط لعددين.

مثال: يكون المُضاعف المُشترك البسيط للعددين 12 و 8 هو 24، ويكون المُضاعف المُشترك البسيط للعددين 20 و 40 هو 80.

التطوير المنهجي للبرمجيات

يمكننا النظر إلى فعل البرمجة على أنه محاولة تحويل الفعل الإنساني إلى فعل آلي تنفذه الآلة.

جرى تطوير مجال واسع من تقنيات التحليل والتصميم البرمجية التي سعت إلى تمكين المبرمجين من التخطيط للكيفية التي ستعمل بها برامجهم قبل البدء بالبرمجة الفعلية.

عموماً، تمر عملية تطوير نظام برمجي بعدة مراحل أهمها:

1. فهم احتياجات المستخدم بدقة ووضوح؛
2. كتابة وصف النظام البرمجي المطلوب (أي توصيفه).
3. استخدام أدوات التصميم والتحليل، بما في ذلك المخططات التدفقية وغيرها.
4. كتابة تعليمات وإجرائيات النظام؛
5. اختبار جميع مكونات وواجهات النظام، وذلك قبل وضعه في حيز التنفيذ الكامل بهدف التحقق من نجاحه وعمله بالشكل المطلوب.
6. إنشاء تطبيق خاص بإعداد وتنصيب النظام، يتضمن الملفات التنفيذية وملحقاتها؛

تَتَّبع عملية تطوير نظامٍ برمجي في عصرنا الحالي منهجية واضحة تعتمد بدايةً على المعرفة والخبرة التي يمتلكها الإنسان عن المشكلة التي سنحلها باستخدام النظام البرمجي، وتسعى في غايتها إلى تحديد الأفعال الدقيقة التي يتوجب على هذا النظام تنفيذها حتى يقدم لنا حلاً للمشكلة المطروحة. لذا يمكننا النظر إلى فعل البرمجة على أنه محاولة تحويل الفعل الإنساني إلى فعل آلي تنفذه الآلة.

تاريخياً، كانت عملية تطوير البرمجيات تعتمد في بداياتها على فعالية المُبرمج وحَدسه التنظيمي الذي كان يساعده في وضع تصور واضح للمشكلة، وفي وضع الخطوات الدقيقة لبناء حلٍ منهجيٍّ لها. إلا أن هذه الأفعال التي كانت تعتمد على الحَدس والتنظيم الشخصي مالبثت أن تحولت إلى آليات منهجية محددة وجرى تطوير مجال واسع من تقنيات التحليل والتصميم البرمجية التي سعت إلى تمكين المبرمجين من

التخطيط للكيفية التي ستعمل بها برامجهم قبل البدء بالبرمجة الفعلية.

عموماً، تمر عملية تطوير نظام برمجي بعدة مراحل أهمها:

1. فهم احتياجات المستخدم بدقة ووضوح؛
2. كتابة وصف النظام البرمجي المطلوب (أي توصيفه). حيث يجري وضع التوصيف في مرحلة تحليل النظام ويتطلب تعاوناً وثيقاً بين محللي النظام من جهة ومستخدميه من جهة أخرى. يتضمن التوصيف شرحاً لكافة عمليات المعالجة التي ينفذها النظام بما فيها:
    - تعريف الدخل؛
    - تعريف الخرج؛
    - الوصف التفصيلي للملفات التي يحتاجها النظام، وبنيتها، والأدوات، والوسائط التي ستُستخدم لتخزينها.
    - الوصف التفصيلي للتقارير، والجداول، والمخططات التي سيجري وضعها.
3. استخدام أدوات التصميم والتحليل، بما في ذلك المخططات التدفقية وغيرها والتي تبين تدفق المعطيات والعلاقات المتبادلة في البرنامج. يجب التنبه خلال هذه المرحلة إلى بعض الأمور الهامة التي ينبغي أخذها بعين الاعتبار:
    - شكل واجهة المستخدم؛
    - نسق ملفات المعطيات؛
    - إمكانية تقسيم البرنامج إلى إجرائيات وواحدات، وإمكانية توزيع العمل على أعضاء فريق البرمجة؛
4. كتابة تعليمات وإجرائيات النظام؛
5. اختبار جميع مكونات وواجهات النظام، قبل وضعه في حيز التنفيذ الكامل بهدف التحقق من نجاحه وعمله بالشكل المطلوب.
6. إنشاء تطبيق خاص بإعداد وتنصيب النظام، يتضمن الملفات التنفيذية وملحقاتها؛

تقنيات التصميم: شبه التشفير

يمكننا اعتماداً على شبه التشفير تحويل خوارزمية إلى شبه برنامج حقيقي وذلك باستخدام تعليمات أساسية تعبر عما يحتاجه المُبرمج من تعليمات أساسية لكتابة البرنامج:

1. تعليمة الدخل: **read**

```
read X;
```

2. تعليمة الخرج: **write**

```
write X;
```

3. عملية إسناد قيمة إلى متحول: **x←5**

```
X←X+5;
```

4. تعليمة الشرط: **if … then begin … end else begin … end**

```
if X>5 then
begin
     write X
end;
```

5. تعليمة تكرار: **while … do begin … end**

```
while X>5 then
begin
    X=X-1
end;
```

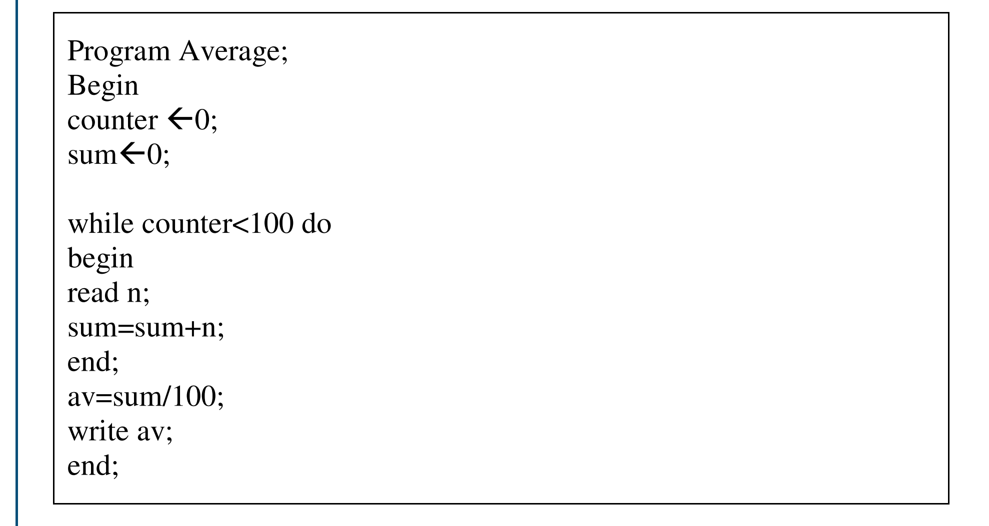
ملاحظات:

- تنتهي كل تعليمة من تعليمات البرنامج بفاصلة منقوطة تعبر عن نهاية التعليمة.
- يبدأ البرنامج بكلمة Program مع إسم البرنامج أو الإجرائية، وتكون تعليماته محاطة بكلمتي begin و end للدلالة على بداية ونهاية البرنامج.
- تُعامَل الإجرائية معاملة البرنامج.
- تُستخدم العمليات الحسابية (+ للجمع، − للطرح، * للضرب، / للقسمة) بين المتحولات؛
- تُستخدم عمليات المقارنة (= يساوي، < أصغر، =< أصغر أو يساوي، > أكبر، => أكبر أو يساوي، =! لايساوي) بين المتحولات.

مثال1: استخدام شبه التشفير لكتابة خوارزمية حساب المتوسط الحسابي لعددين يُدخلهما المُستخدِم

```
Program Average;
begin
read n1;
read n2;
av←(n1+n2)/2;
write av;
end;
```

مثال2: استخدام شبه التشفير لكتابة خوارزمية حساب المتوسط الحسابي لــ 100 عدد يُدخلها المُستخدِم



```
Program Average;
Begin
counter ←0;
sum←0;

while counter<100 do
begin
read n;
sum=sum+n;
end;
av=sum/100;
write av;
end;
```

هناك العديد من تقنيات التصميم التي تُستخدم على عدة مستويات، لكننا سنركز في هذا الفصل على شبه التشفير كونه يُعتبر مرحلة انتقالية بين التوصيف المبدئي، وبين عملية البرمجة الفعلية للبرنامج.

تبدو عملية شبه التشفير مشابهة جداً لتعليمات البرنامج التي ستجري كتابتها فعلياً، حيث يجري استخدام عبارات شبيهة بالعبارات البرمجية، مثل: IF و WHILE، ويجري تحديد تعليمات البرمجة المطلوبة، دون الانتباه للقواعد الحرفية للغة البرمجية المستخدمة أو لاستكمال كافة إجرائيات البرنامج.

يمكننا اعتماداً على شبه التشفير تحويل خوارزمية إلى شبه برنامج حقيقي وذلك باستخدام تعليمات أساسية تعبر عما يحتاجه المُبرمج من تعليمات أساسية لكتابة البرنامج:

1. تعليمة الدخل: **read**
2. تعليمة الخرج: **write**
3. عملية إسناد قيمة إلى متحول: **x←5**.
4. تعليمة الشرط: **if … then begin … end else begin … end**
5. تعليمة تكرار: **while … do begin … end**

ملاحظات:

- تنتهي كل تعليمة من تعليمات البرنامج بفاصلة منقوطة تعبر عن نهاية التعليمة.
- يبدأ البرنامج بكلمة Program مع إسم البرنامج أو الإجرائية، وتكون تعليماته محاطة بكلمتي begin و end للدلالة على بداية ونهاية البرنامج.
- تُعامَل الإجرائية معاملة البرنامج.
- تُستخدم العمليات الحسابية (+ للجمع، - للطرح، * للضرب، / للقسمة) بين المتحولات؛
- تُستخدم عمليات المقارنة (= يساوي، > أصغر، =< أصغر أو يساوي، < أكبر، => أكبر أو يساوي، =! لايساوي) بين المتحولات.

نشاطات للمناقشة والحلّ خلال الجلسات

اكتب الخوارزميات التالية **باستخدام شبه التشفير**:

أولاً– خوارزمية إدخال عددين (الأول أكبر من الثاني) واحتساب ما يمثله الثاني كنسبة مئوية من الأول.

ثانياً– خوارزمية حساب المتوسط الحسابي لعدد غير محدود من الأرقام الموجبة يقوم المُستخدم بإدخالها ويعلن عن إنتهائه من إدخال الأرقام بإدخال رقم سالب.

ثالثاً –خوارزمية حساب المُضاعف المُشتَرَك البسيط لعددين.

استراتيجيات وضع الحلول البرمجية

تتضمن استراتيجية وضع حلّ برمجي:

- التعرف على المشكلة وتأطيرها؛
- وضع تصميم للحلّ المُقتَرح؛
- تنفيذ مجموعة من الاختبارات على الحلّ؛
- التوثيق.

تتضمن استراتيجية وضع حلّ برمجي:

التعرف على المشكلة وتأطيرها:

تتطلب هذه المرحلة فهماً كاملاً للمشكلة، بحيث يمكن تعريف الافتراضات التي يمكن استخدامها والافتراضات غير الممكنة، وذلك بهدف اختبار الحل على النحو الملائم. فعلى سبيل المثال، عند كتابتنا لبرنامج حساب المتوسط الحسابي لمجموعة من الأرقام تبدو المشكلة واضحة نسبياً حين وضع بعض الافتراضات على الأرقام التي ينبغي إدخالها مثل:

- تحديد نمط الأرقام (صحيحة، حقيقية)؛
- ضرورة إظهار رسالة خطأ في حال أدخل المُستخدم محرفاً آخر (حرف ما)؛
- تحديد نوع الأرقام الصحيحة (سالبة، موجبة أم مختلطة).

لذا، لا بد من القيام بتحليل شامل لفهم المشكلة تماماً بحيث يمكن صياغة تعريف كامل بالفرضيات عند انتهاء التحليل. وتتضمن

الفرضيات، بالإضافة للحالات التي ذكرناها في مثالنا السابق:

- اللغة البرمجية التي ينبغي استخدامها؛
- كمية المعطيات التي يتوجب معالجتها.

وضع تصميم للحلّ المُقترح:

نحتاج بعد تحديد متطلبات البرنامج وتأطير المشكلة التي يعالجها، إلى تحديد خطوطه الرئيسية اللازمة لتنفيذ وتحقيق المتطلبات. لذا يجري استخدام أدوات ومنهجيات متعددة كالخوارزميات، والمخططات التدفقية وأساليب التحليل من القمة إلى القاعدة لتوصيف خطوط الحلّ.

ويعتبر أسلوب التصميم من القمة إلى القاعدة أسلوباً منهجياً لتحليل المشكلات باستخدام مبدأ الوحدات. ومن المتعارف عليه هو أن وضع تصميم باستخدام أسلوب التصميم من القمة إلى القاعدة، ومبادئ البرمجة المهيكلة يعطي برامجاً واضحة، وصحيحة، وسهلة الاختبار، والصيانة.

تنفيذ مجموعة من الاختبارات على الحلّ:

ويشمل استخدام معطيات الاختبار للتحقق يدوياً من تعليمات البرنامج، بحيث يمكن مقارنة النتائج المتوقعة منها مع النتائج الفعلية التي يقوم البرنامج بتنفيذها.

التوثيق:

يجب أن يتضمن توثيق البرنامج:

- توصيف النظام وتوصيف الدخل والخرج، والخطوط العريضة لأقسامه؛
- تصميم البرنامج، بما في ذلك بنى المعطيات والخوارزميات وشبه التشفير؛
- البرنامج النهائي بالتفصيل، ويتضمن خطة الاختبارات، وسجلات الاختبار.

ويعد التوثيق أمراً أساسياً للأسباب التالية:

- لإعطاء المستخدم فكرة عن كيفية إعداد معطيات البرنامج وطريقة تفسير الخرج؛
- لتسهيل عملية صيانة البرنامج،
- لتسهيل التعديلات المستقبلية التي تهدف إلى تطوير البرنامج.

التصميم من القمة إلى القاعدة

بالإمكان التوصل إلى حل لمعظم المشاكل بعد معرفتها وتفهمها. ومن المناهج الشائعة في ذلك منهج التصميم من القمة إلى القاعدة الذي يُعرّف عملية معالجة عامة تساعد في الإنطلاق من مشكلة كبيرة معقدة إلى مجموعة من المشكلات الأصغر، بحيث تكون قابلية حل هذه الأجزاء أسهل من حل المشكلة الأصلية دفعة واحدة. هذا وتتضمن عملية التصميم بمجملها كل مما يلي:

- وضع توصيف مفصل يحدد دخل وخرج النظام والفرضيات الأساسية الخاصة به؛
- كتابة شبه شيفرة الخاص بكل قسم من أقسام النظام البرمجي؛
- ترجمة شبه الشيفرة إلى لغة برمجية.

مثال:

نحتاج لكتابة برنامج لحل معادلات من الدرجة الثانية.

بما أن معادلة من الدرجة الثانية تُكتب على الشكل $ax^2 + bx + c = 0$، لذا فإن حلها يتطلب معرفة 3 عناصر هي: a، و b، و c. إذاً:

دخل البرنامج: ثلاث أعداد حقيقية هي a، b، c تمثل أمثال حدود المعادلة.

بما أن المطلوب هو حلّ المعادلة، فمن الممكن أن يكون للمعادلة حلّ وحيد، أو حلين، أو لايكون لها حلول. إذاً:

خرج البرنامج: إحدى الحالات التالية:

1. **رسالة تعلن عدم وجود حلّ؛**
2. **رسالة تعلن عن وجود حل وحيد مع إظهار الحلّ؛**
3. **رسالة تعلن عن وجود حلين مع إظهار الحلّين.**

لنسأل نفسنا الآن عن الفرضيات التي يجب معالجتها، والحقيقة أنه لا توجد أية فرضيات على قيم a و b و c التي يمكن أن تكون حقيقية أو صحيحة، سالبة أو موجبة.

بالنتيجة تكون خوارزمية الحل بشبه التشفير كما يلي:

```
program equation;
begin
        read a;
        read b;
        read c;

        delta = b^2 - 4×a×c;

        if delta<0 then
        begin
                write "No Solution";
        end;

        if delta = 0 then
        begin
                write "One Solution:";
                sol = -b / (2×a);
                write sol;
        end;

        if delta >0 then
        begin
                write "Two Solutions:";
                sol1 = (-b + √delta) / (2×a);
                sol2 = (-b - √delta) / (2×a);
                write sol1,sol2;
        end;
end;
```

نشاط

المسألة الأولى:

نريد كتابة برنامج حساب المتوسط الحسابي:

$$Average = \frac{\sum_{i=1}^{n} x_i}{n}$$

حيث تعبّر x_i عن معدلات طلاب صف من صفوف الجامعة الافتراضية. مع العلم أن عدد طلاب الصف الواحد (المُشار إليه بالمتحول n) يبلغ 25 طالباً، وأن المعدلات محسوبة من 100 علامة وأن المُستخدم يقوم بإدخال المعدلات عند تنفيذ البرنامج.

حدد عند توصيفك ومعالجتك للمطلوب:

1- دخل كل برنامج؛
2- خرج كل برنامج؛
3- الفرضيات الأساسية التي يجب معالجتها في كلا البرنامجين؛
4- الخوارزمية الخاصة بكل برنامج على شكل شبه تشفير؛

المسألة الثانية:

نريد كتابة برنامج حساب الإنحراف المعياري:

$$StandardDeviation = \frac{\sum_{i=1}^{n}(x_i - Average)}{n}$$

حيث تعبّر x_i عن معدلات طلاب صف من صفوف الجامعة الافتراضية، ويعبر Average عن المتوسط الحسابي للمعدلات. مع العلم أن عدد طلاب الصف الواحد (المُشار إليه بالمتحول n) يبلغ 25 طالباً وأن المعدلات محسوبة من 100 علامة وأن المُستخدم يقوم بإدخال المعدلات عند تنفيذ البرنامج.

حدد عند توصيفك ومعالجتك للمطلوب:

1- دخل كل برنامج؛
2- خرج كل برنامج؛
3- الفرضيات الأساسية التي يجب معالجتها في كلا البرنامجين؛
4- الخوارزمية الخاصة بكل برنامج على شكل شبه تشفير؛

المسألة الثالثة:

بفرض أن لديك معادلة من الدرجة الأولى:

$$ax + b = c$$

حيث تعبر a,b,c عن قيم صحيحة أوحقيقية، في حين يعبر x عن متحول.

المطلوب

1. وضع الحل الرياضي لمعادلة من الدرجة الأولى؛
2. كتابة خوارزمية حل معادلة من الدرجة الأولى بحيث تحدد:
   a. دخل البرنامج؛

b. خرج البرنامج؛
c. الفرضيات الأساسية التي يجب معالجتها في البرنامج؛
d. الخوارزمية الخاصة بالبرنامج على شكل شبه تشفير.

المسألة الرابعة:

بفرض أن لديك معادلة مستقيم:

$$ax + by = c$$

تعبر a,b,c عن قيم صحيحة أوحقيقية، في حين يعبر كل من x و y عن متحولين، إذ يعبر x عن محور الفواصل (المحور OX) ويكون y هو المتحول المعبّر عن محور التراتيب (المحور OY).

نقول عن نقطة (x_0,y_0) أنها منتمية إلى المستقيم ax+by=c إذا تحققت معادلة المستقيم بتعويض x و y بـــ x_0 وy_0 على الترتيب، كمايلي:

$$a \times x_0 + b \times y_0 = c$$

فعلى سبيل المثال، ومن أجل المستقيم $5x + 4y = 13$ تكون النقطة (1,2) منتمية إلى المستقيم لأن: $5\times1+4\times2=13$

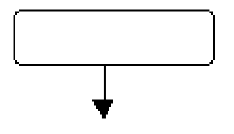
المطلوب كتابة خوارزمية التحقق من انتماء نقطة إلى مستقيم بحيث تحدد:

e. دخل البرنامج (**مساعدة:** المستقيم والنقطة)؛
f. خرج البرنامج (**مساعدة:** انتماء أو عدم انتماء)؛
g. الفرضيات الأساسية التي يجب معالجتها في البرنامج؛
h. الخوارزمية الخاصة بالبرنامج على شكل شبه تشفير.

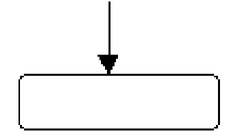
المخططات التدفقية

تُعتَبَر المخططات التدفقية إحدى أدوات التصميم المرئي للأنظمة البرمجية وخصوصاً الصغير منها. وتُستَخدِم المخططات التدفقية رموزاً خاصة بها نستعرضها فيما يلي:

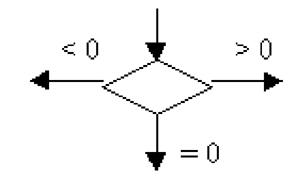
1. البداية:



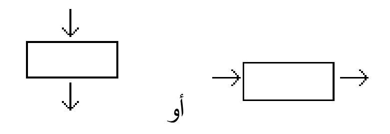
2. النهاية:



3. الاقتضاء الشرطي:



4. العمليات:



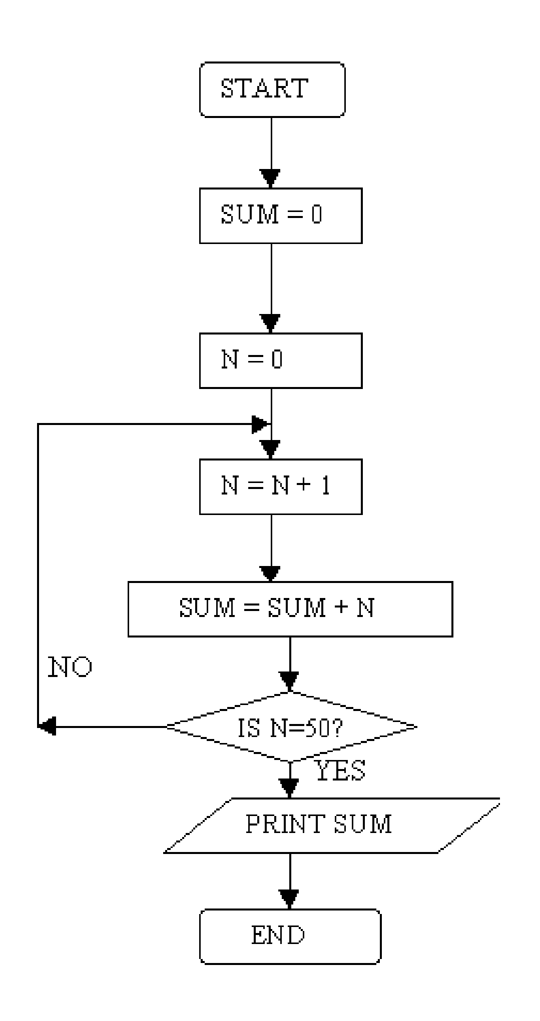
5. الدخل والخرج:



مثال1:

صمم المخطط التدفقي الذي يمثل برنامجاً يساعد في حساب مجموع الأعداد من 1 إلى 50.

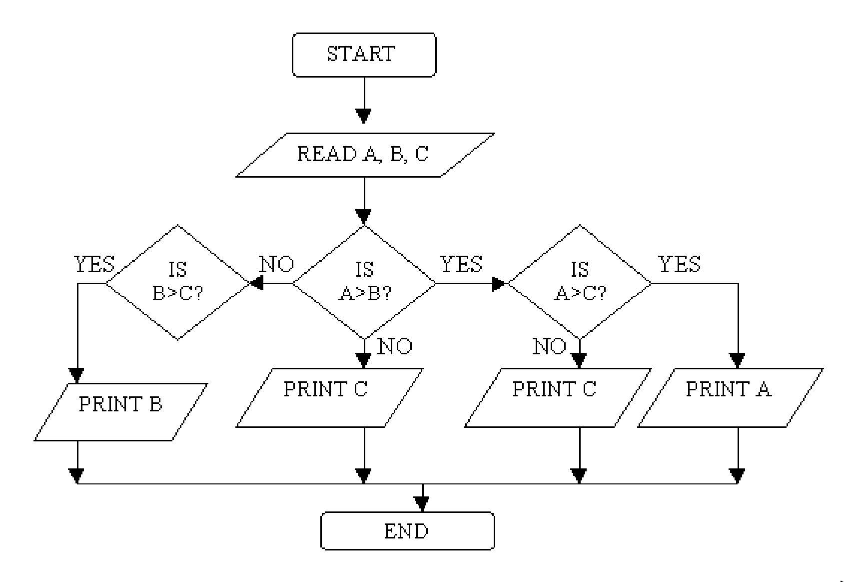
الحلّ:



مثال2:

صمم المخطط التدفقي الذي يمثل برنامجاً يساعد في إيجاد العدد الأكبر من بين ثلاثة أعداد A, B, C يُدخلها المُستخدِم.

الحلّ:

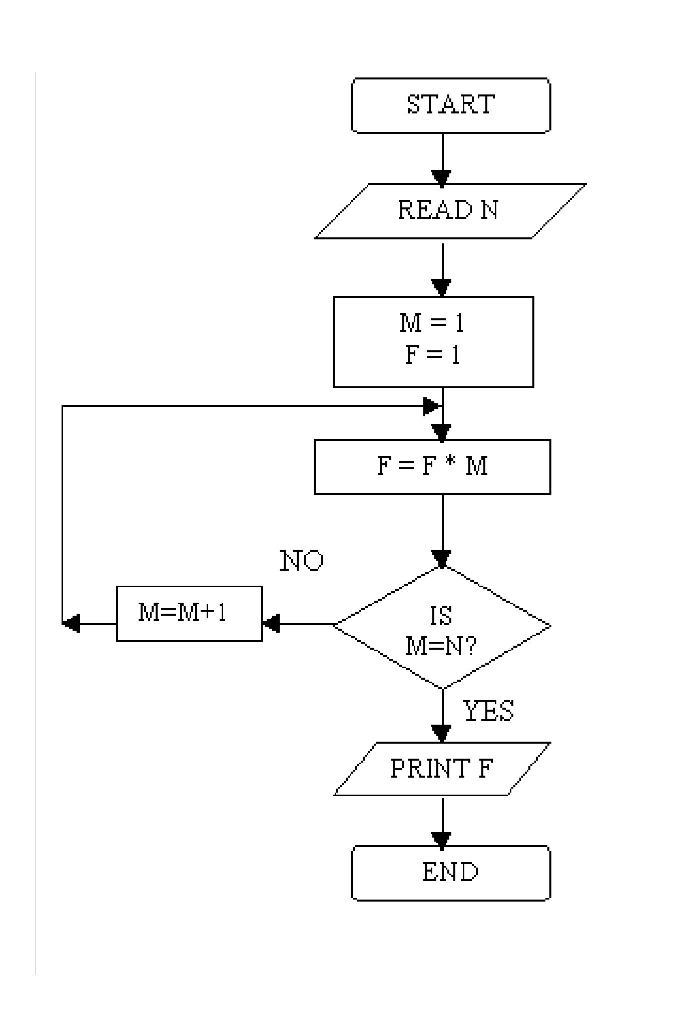


مثال3:

صمم المخطط التدفقي الذي يمثل برنامجاً يساعد في حساب **N!** (N عاملي) حيث N هو عدد يُدخله المُستخدِم.

(N! = N * (N-1) * (N-2) * (N-3) * … * 3 * 2 * 1)

الحلّ:



نشاطات للمناقشة والحلّ خلال الجلسات

صمم المخطط التدفقي الخاص بكل برنامج من البرامج التالية:

1. برنامج يقرأ عدد صحيح N ويعطي جميع الأعداد التي يقبل N القسمة عليها (بحيث يكون ناتج القسمة عدد صحيح).
   مساعدة: يمكنك أن تفترض أن لديك اختبار يساعدك على معرفة فيما إذا كان عدد k صحيح أم لا وهو الاختبار isInteger(k).

2. برنامج يقرأ 5 أعداد ويعطي مجموع مربعاتها.
   مساعدة: يمكنك أن تفترض أن لديك تابع لحساب مربع عدد k هو التابع sqr(k).

3. برنامج يسمح لك بإدخال رقمين واحتساب ما يمثله الرقم الأصغر كنسبة مئوية من الرقم الأكبر.

4. برنامج يسمح بحساب القاسم المشترك الأعظم لعددين صحيحين. مثال: القاسم المشترك الأعظم للعددين 6 و 12 هو 3. والقاسم المشترك الأعظم للعددين 24 و20 هو 4.
   مساعدة: يمكنك أن تفترض أن لديك اختبار يساعدك على معرفة فيما إذا كان عدد k صحيح أم لا وهو الاختبار isInteger(k).

أسئلة

أجب بصح أو خطأ:

1. ترتبط البرامج المكتوبة بلغات عالية المستوى بالعتاد؛ خطأ
2. تكون كتابة البرامج بلغات عالية المستوى أصعب من كتابتها بلغات ذات مستوى منخفض؛ خطأ
3. يكون تشغيل البرامج المكتوبة بلغات عالية المستوى أبطأ من البرامج المكتوبة بلغات ذات مستوى منخفض؛ صح
4. يكون تصحيح البرامج المكتوبة بلغات عالية المستوى أسهل من البرامج المكتوبة بلغات ذات مستوى منخفض؛ صح
5. يتحكم نظام التشغيل بالموارد؛ صح
6. إن نظام التشغيل هو Windows؛ خطأ
7. يعد نظام التشغيل جزءاً من العتاد؛ خطأ
8. يمكن تحميل أكثر من برنامج في الذاكرة في آن واحد؛ صح
9. يمكن تنفيذ وتشغيل أكثر من برنامج في آن واحد؛ صح
10. يمكن أن تستخدم عدة برامج الطابعة في آن واحد. صح

رتب ما يلي وفق التتالي الزمني الصحيح:

1. الاختبارات؛ (5)
2. البرمجة؛ (4)
3. كتابة توصيف البرنامج؛ (2)
4. التوثيق؛ (6)
5. التصميم؛ (3)
6. فهم المشكلة؛ (1)

القسم الخامس والسادس

أساسيات لغة #C

الكلمات المفتاحية:

فضاء الأسماء، الأنماط المُنمَّرة، صفّ، ثابت، متحول،

ملخص:

نتعرف في هذا القسم على أساسيات لغة البرمجة #C؛

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- تشغيل محيط التطوير Dot Net بهدف برمجة تطبيقات بسيطة بلغة #C؛
- الأنماط الأساسية؛
- الأنماط المُنمَّرة؛
- المتحولات والثوابت؛
- العمليات الحسابية والمنطقية وعمليات المقارنة؛
- أفضليات العمليات الأساسية؛

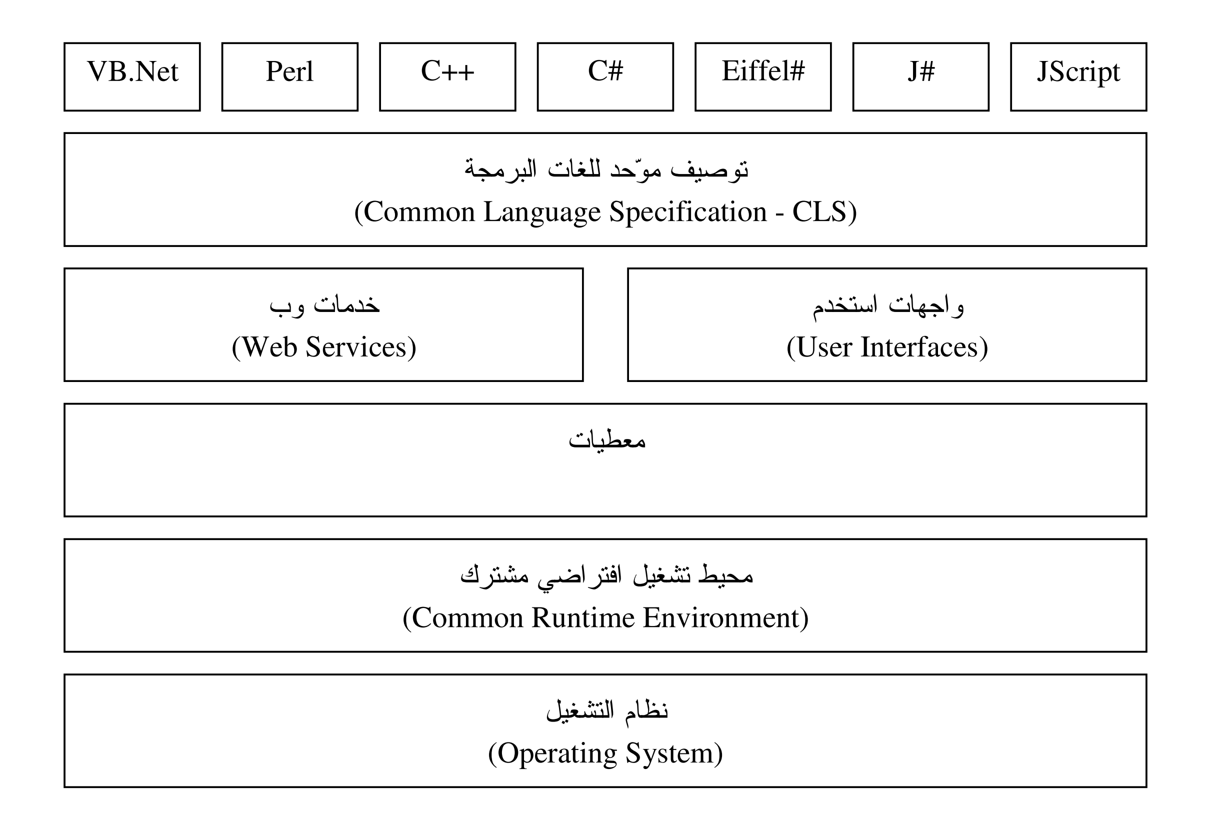
Microsoft Dot Net

اقترحت Microsoft استراتيجية جديدة لتوزيع عملية معالجة المعطيات في إطار بنيان برمجي متكامل، تحت اسم "Dot Net". يرتكز البنيان الآنف الذكر على مجموعة من الأفكار المؤسِسة التي تطمح للوصول إلى بيئة برمجية تتمتع بالمواصفات التالية:

- شفافية التعامل مع التطبيق من ناحية كونه تطبيق محلي أو تطبيق إنترنت؛
- توزيع المعطيات على عدد من المخدمات عوضاً عن تركيزها ضمن مخدم واحد، وبحيث يحتوي كل مخدم على الخدمات اللازمة للتعامل مع جزئه الخاص من المعطيات؛
- تحويل عملية شراء تطبيق برمجي وتثبيته على مخدمات محلية إلى عملية استئجار خدمة برمجية تقدمها مجموعة مخدمات على الإنترنت؛
- تحويل الحاسب الشخصي إلى طرفية ذكية تساعد في البحث عن الخدمة المطلوبة وتشغيلها عن بعد؛
- تأمين مكونات برمجية جاهزة يمكن لمطوري البرامج مكاملتها ضمن برامجهم دون الحاجة لإعادة برمجتها؛

بالنتيجة، تقدم Microsoft من خلال Dot Net محيطاً برمجياً يُدعى (Dot Net Framework) يساعد المُبرمج في برمجة وتشغيل تطبيقاته سواءاً كانت تطبيقات كلاسيكية تعمل ضمن محيط نظام التشغيل Windows أو تطبيقات وب تعمل ضمن محيط مخدم وب.

بنيان إطار العمل Dot Net Framework



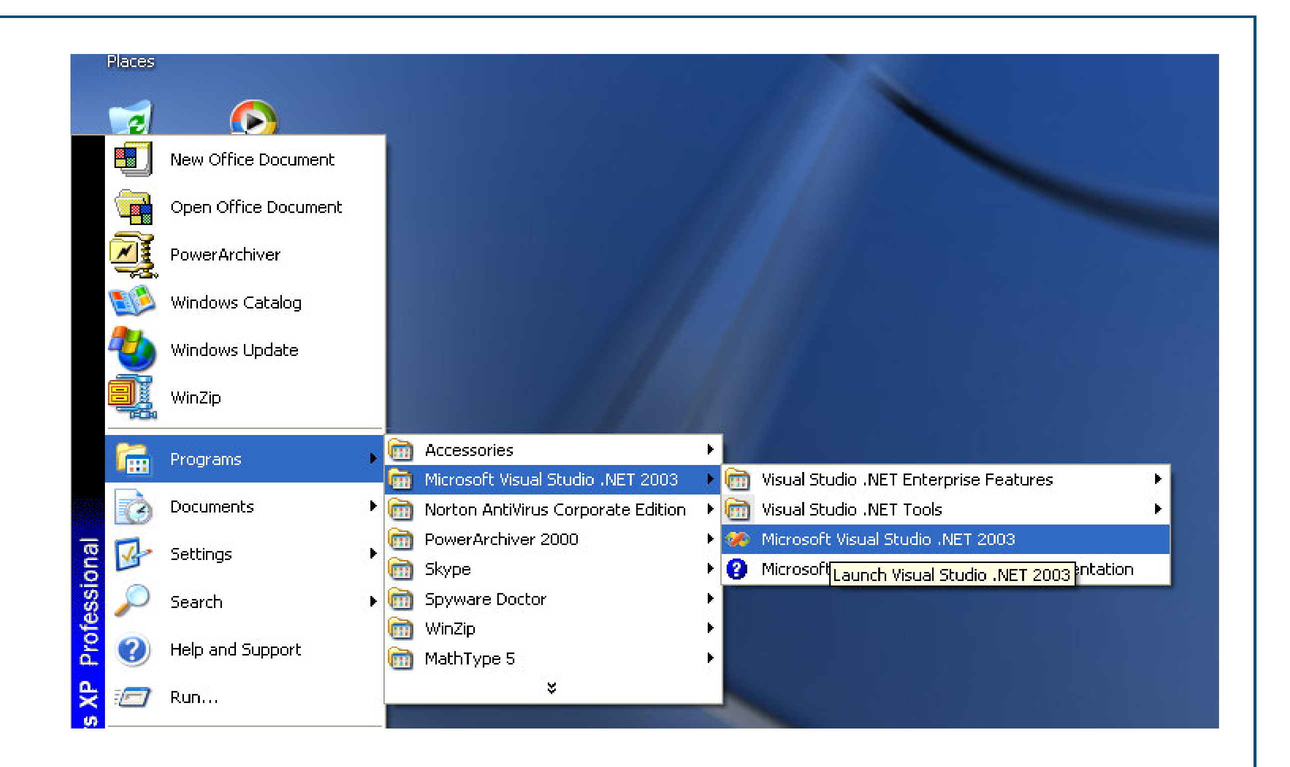
يُقسم ينيان إطار العمل Dot Net إلى مجموعة من الطبقات التي تساعد المُبرمج على كتابة برامجه وتحويلها إلى برامج تنفيذية.

سنستعرض في هذا الشكل مجموعة الطبقات وسنركز في بقية القسم على عمل الطبقتين الأولى والثانية التي تهمنا كمبرمجين وخصوصاً بالنسبة للغة C# وأسلوب استثمارها لهذا المحيط.

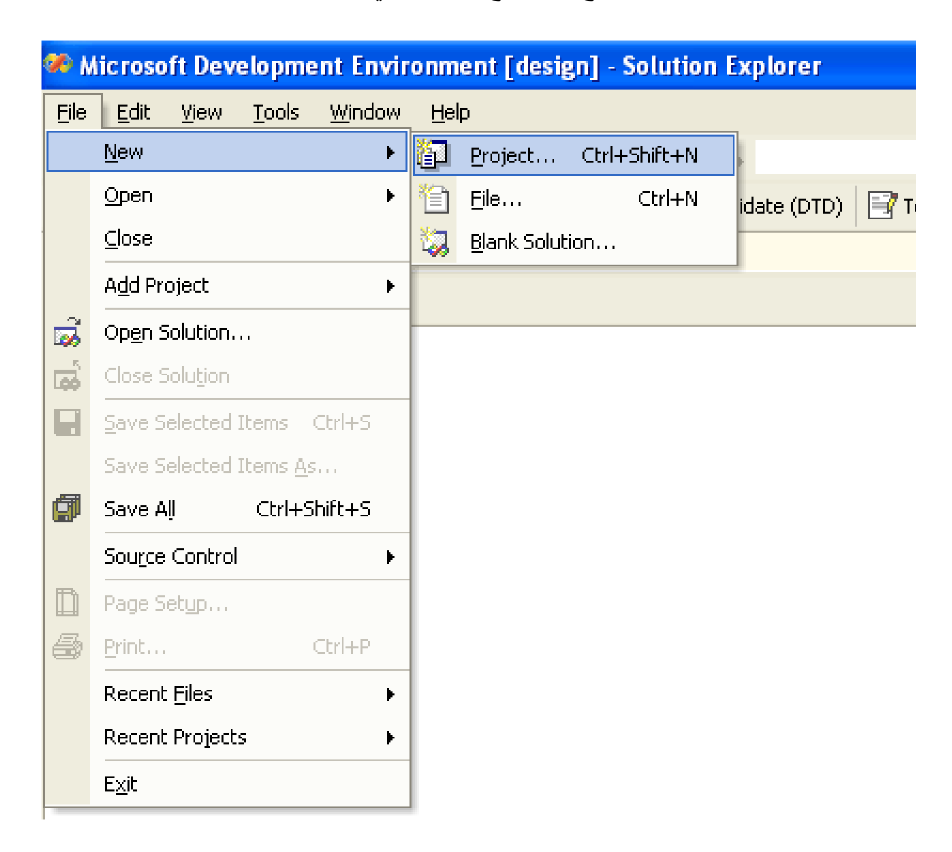
بداية سريعة مع C#

بعد تثبيت Visual Studio.net نفذ العمليات التالية:

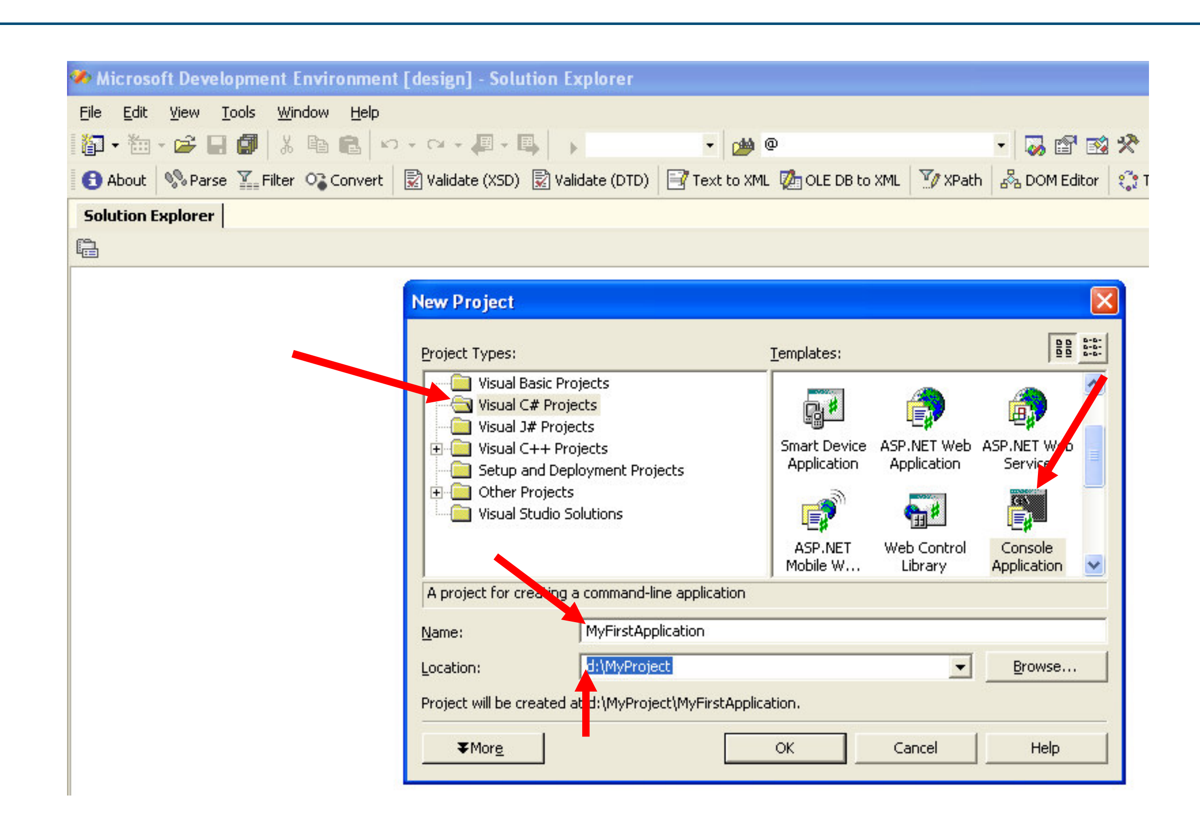
1. إذهب إلى زر البداية Start وإلى مكان إقلاع Microsoft Visual Studio .Net:



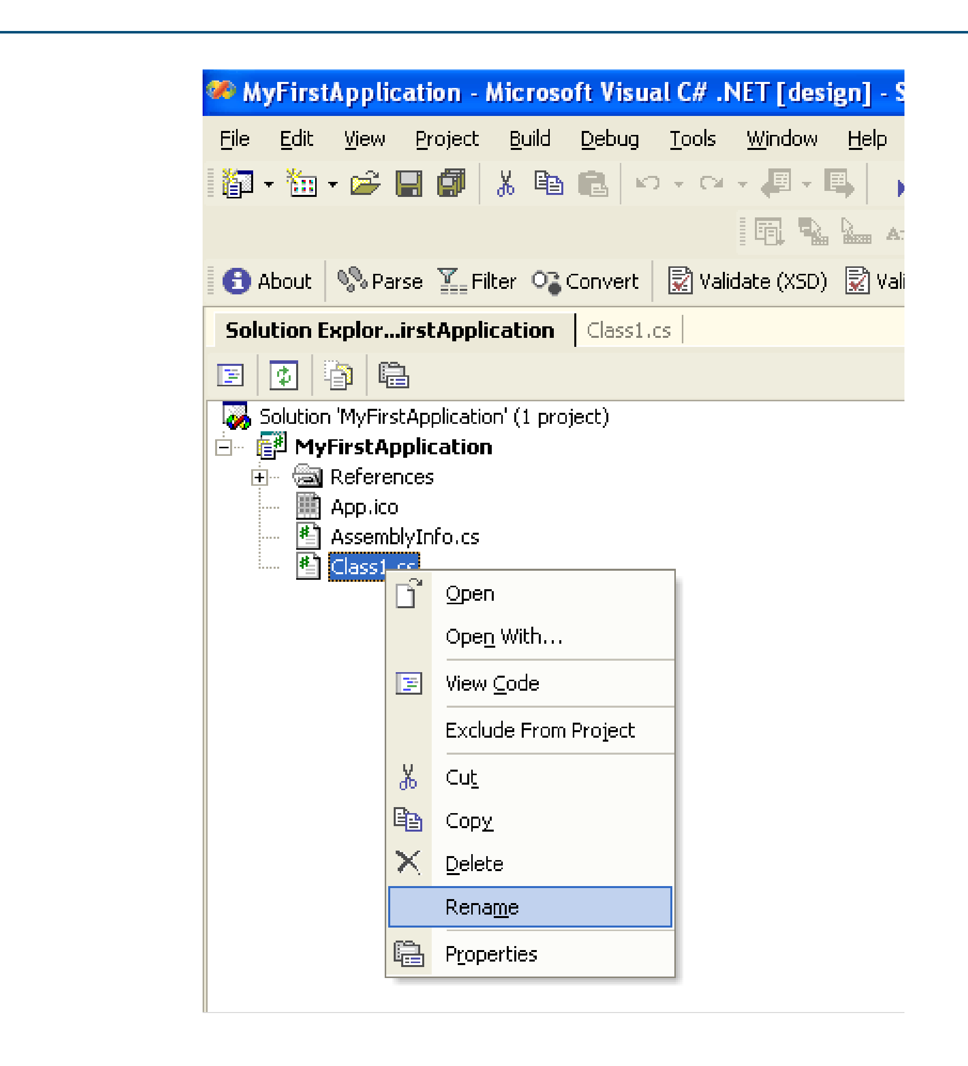
2. عند إقلاع Visual studio .Net إذهب إلى نافذة فتح المشاريع الظاهرة في الشكل:



3. يمكنك عندها اختيار C# Project من النافذة اليسارية، و Consol Application من النافذة اليمينية مع تحديد اسم التطبيق ومكان تخزينه في الأسفل، كما هو موضح في الشكل:



4. عند حصولك على واجهة التطبيق، اضغط بالزر اليميني على الصف class1.cs لتسميته بالإسم الذي تريد، وليكن مثلاً Hello.cs، ثم اضغط مرة أخرى بالزر اليميني، واذهب إلى واجهة View Code:



5. ستحصل على الواجهة التالية التي تؤطر البرنامج الذي ستكتبه:

MyFirstApplication - Microsoft Visual C# .NET [design] - Class1.cs

File Edit View Project Build Debug Tools Window Help

Debug

About Parse Filter Convert Validate (XSD) Validate (DTD) Text to XML OLE

Solution Explorer - MyFirstApplication | Class1.cs

MyFirstApplication.Class1 | Main(s

Types

```
using System;

namespace MyFirstApplication
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
    class Class1
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main(string[] args)
        {
            //
            // TODO: Add code to start application here
            //
        }
    }
}
```

6. يمكنك كتابة برنامجك الأول الموضح فيما يلي ضمن إطار النص البرمجي المُعطى:

```
using System;

namespace MyFirstApplication
{
        /// <summary>
        /// I would like to say Hello Universe
        /// </summary>
        class Hello
        {
                /// <summary>
                /// The main entry point for the application.
                /// </summary>
                public static void Main(string[] args)
                        {
                                Console.WriteLine("Hello, Universe");
                                // Iterate over command line arguments,
                                // and print them out
                                for (int arg = 0; arg < args.Length; arg++)
                                        Console.WriteLine("Arg {0}: {1}", arg, args[arg]);
                        }
        }
}
```

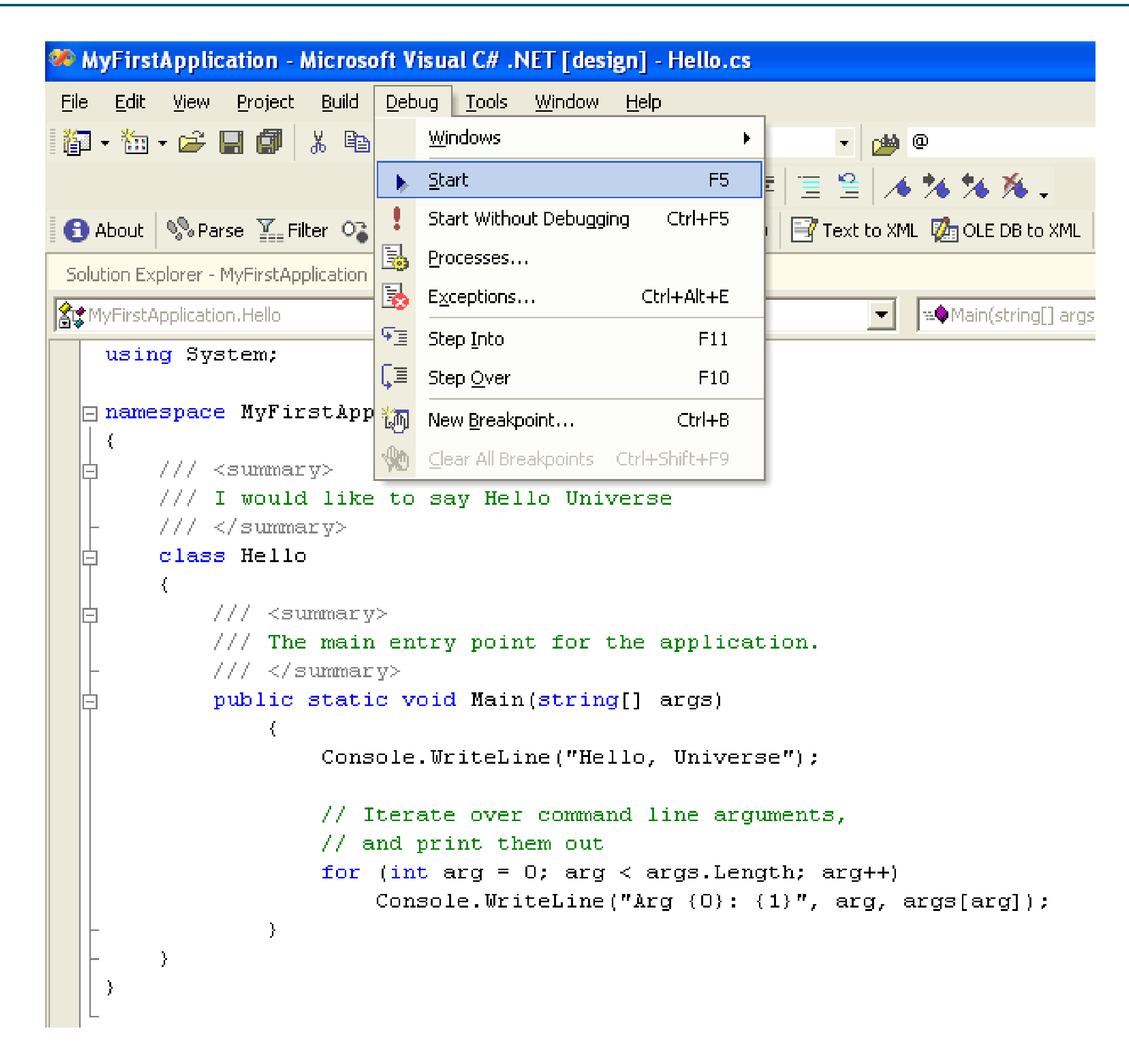
MyFirstApplication - Microsoft Visual C# .NET [design] - Hello.cs

```
using System;

namespace MyFirstApplication
{
    /// <summary>
    /// I would like to say Hello Universe
    /// </summary>
    class Hello
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        public static void Main(string[] args)
            {
                Console.WriteLine("Hello, Universe");

                // Iterate over command line arguments,
                // and print them out
                for (int arg = 0; arg < args.Length; arg++)
                    Console.WriteLine("Arg {0}: {1}", arg, args[arg]);
            }
    }
}
```

7. يمكنك ترجمة برنامجك والتحقق من صحته بالذهاب إلى واجهة التنفيذ Debug ومن ثم Start:



8. يمكن بعد ذلك تنفيذ البرنامج اعتباراً من Dos Command Prompt كما يظهر من الشكل تحت اسم MyFirstApplication.exe:

```
Command Prompt
09/27/2006  09:27 AM             4,015 MyFirstApplication.csproj
09/27/2006  09:35 AM             1,803 MyFirstApplication.csproj.user
09/27/2006  09:35 AM               921 MyFirstApplication.sln
09/27/2006  09:27 AM    <DIR>          obj
               6 File(s)         10,755 bytes
               4 Dir(s)   1,563,451,392 bytes free

D:\MyProject\MyFirstApplication>cd bin

D:\MyProject\MyFirstApplication\bin>cd debug

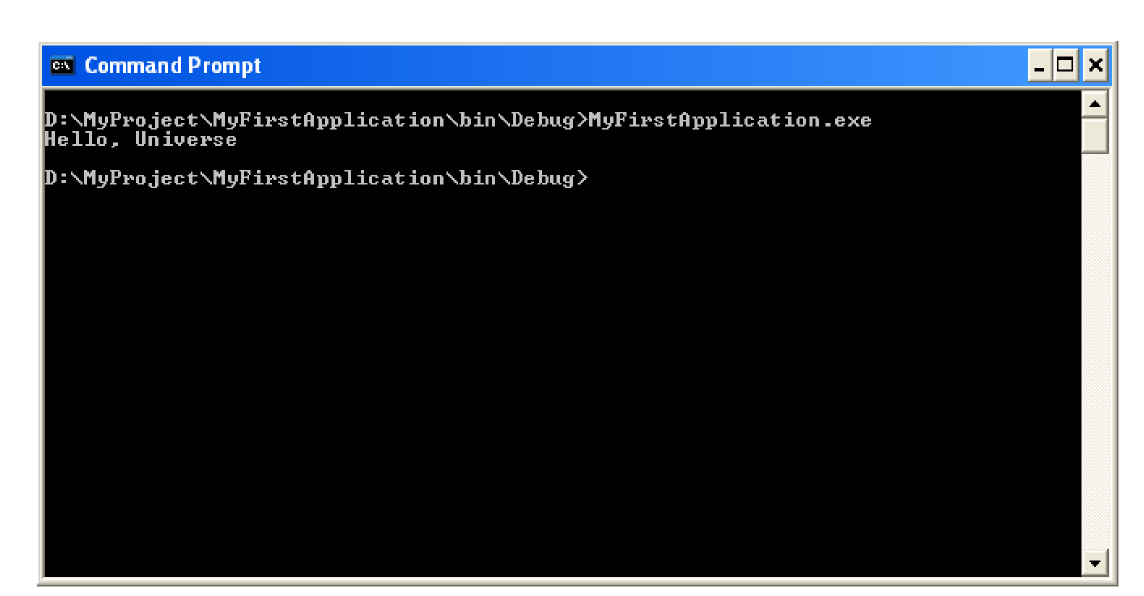
D:\MyProject\MyFirstApplication\bin\Debug>dir
 Volume in drive D is Data
 Volume Serial Number is D836-C733

 Directory of D:\MyProject\MyFirstApplication\bin\Debug

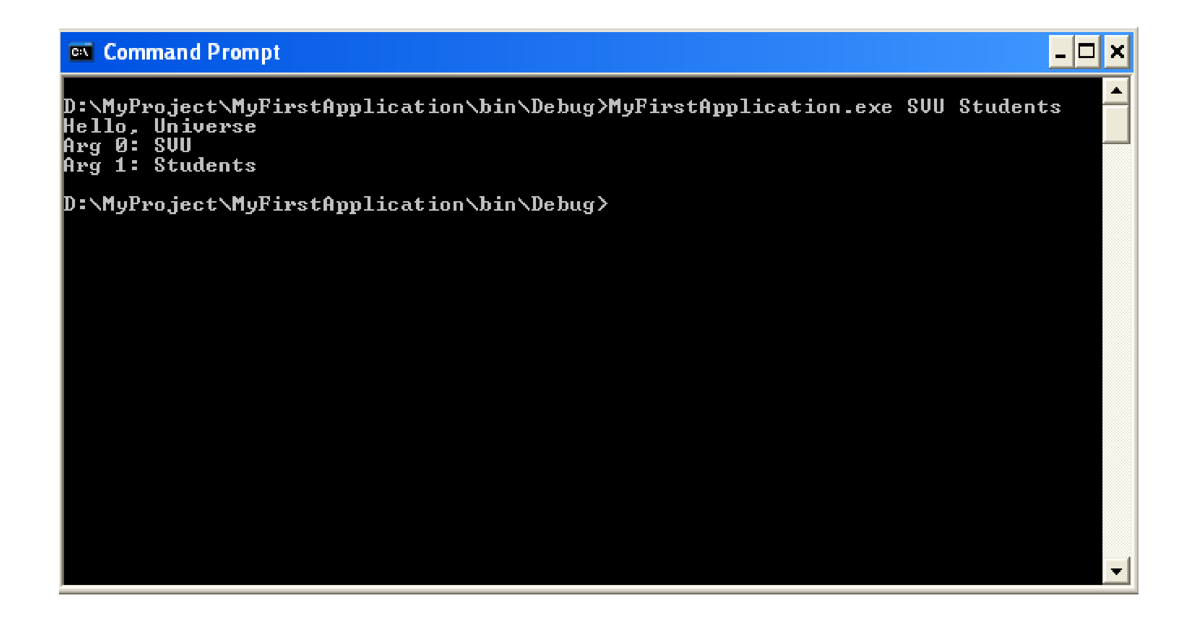
09/27/2006  09:35 AM    <DIR>          .
09/27/2006  09:35 AM    <DIR>          ..
09/27/2006  09:35 AM            16,384 MyFirstApplication.exe
09/27/2006  09:35 AM            11,776 MyFirstApplication.pdb
               2 File(s)         28,160 bytes
               2 Dir(s)   1,563,451,392 bytes free

D:\MyProject\MyFirstApplication\bin\Debug>_
```

9. يمكن بعد ذلك تنفيذ البرنامج بدون مُعاملات:



أو مع معاملات:



تحليل النص البرمجي

```
using System;

namespace MyFirstApplication
{
	/// <summary>
	/// I would like to say Hello Universe
	/// </summary>
	class Hello
	{
		/// <summary>
		/// The main entry point for the application.
		/// </summary>
		public static void Main(string[] args)
			{
				Console.WriteLine("Hello, Universe");

				// Iterate over command line arguments,
				// and print them out
				for (int arg = 0; arg < args.Length; arg++)
					Console.WriteLine("Arg {0}: {1}", arg,
args[arg]);
			}
	}
}
```

- تُستَخدَم العبارة "using system" للدلالة على ماندعوه فضاء الأسماء "system" الذي يُقدم مجموعة من الصفوف الجاهزة والمُعرّفة التي يمكن استخدامها ضمن التطبيق مباشرةً؛
- ينتمي الصف "Console" إلى فضاء الأسماء "system" ويُستَخدَم للتعامل مع واجهة التعليمات النصيّة التي ندعوها "Command Prompt" كما لاحظنا عند تشغيل البرنامج.
- يستخدم الصف "Console" الإجرائية "Writeline" لكتابة سلسلة محارف وإظهارها على واجهة التعليمات النصيّة.
- يُعرِّف المثال صفاً يُدعى "Hello"، يحتوي على إجرائية "Main" يمكن اعتبارها نقطة إنطلاق لتنفيذ المثال؛
- يقوم الإجرائية بإظهار عبارة "Hello Universe" بالإضافة إلى أية عبارة أخرى يُدخلها المُستثمر عند استدعائه للتطبيق من واجهة التعليمات النصيّة.

الأنماط الأساسية

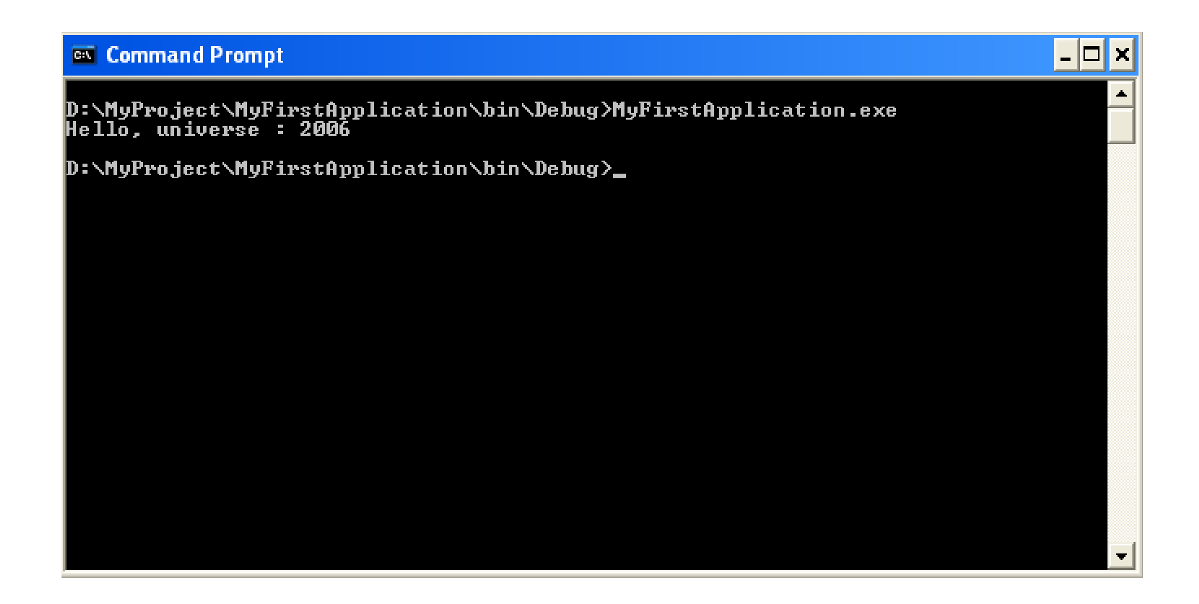
يمكن للمُبرمج استخدام أحد الأنماط البسيطة التي تظهر في الجدول لتعريف متحولاته. ويؤدي تعريف كل نوع من أنواع المتحولات الظاهر حجز جزء من الذاكرة يُقدر بالبايت ويتعلق بالنمط المُستَخدَم.

النمط	عدد الـ Byte التي يحجزها في الذاكرة	توصيف
byte	1	قيمة صحيحة بدون إشارة تتراوح بين 0 و 255
sbyte	1	قيمة صحيحة ذات إشارة تتراوح بين -128 و +127
short	2	قيمة صحيحة ذات إشارة تتراوح بين $2^{-15}-1$ و $+2^{15}$
ushort	2	قيمة صحيحة بدون إشارة تتراوح بين 0 و $2^{16}-1$
int	4	قيمة صحيحة ذات إشارة تتراوح بين $2^{-31}-1$ و $+2^{31}$
uint	4	قيمة صحيحة بدون إشارة تتراوح بين 0 و $2^{32}-1$
long	8	قيمة صحيحة ذات إشارة تتراوح بين $2^{-63}-1$ و $+2^{63}$
ulong	8	قيمة صحيحة بدون إشارة تتراوح بين 0 و $2^{64}-1$
float	4	فاصلة عائمة بدقة بسيطة ~ 7 أرقام عشرية
double	8	فاصلة عائمة بدقة مُضاعفة ~ 15 رقم عشري
decimal	16	دقة تصل إلى 28 رقم عشري على الأكثر
string		سلسلة محارف
char	16	حرف يتبع الترميز Unicode (قيمته بين 0 و 65536)
bool		يأخذ إحدى القيمتين TRUE أو FALSE

مثال:

```
using System;
namespace MyFirstApplication
{
	/// <summary>
	/// I would like to say Hello Universe
	/// </summary>
	class Hello2
	{
		/// <summary>
		/// The main entry point for the application.
		/// </summary>
		public static void Main(string[] args)
		{
			int Y=2006;
			string s="Hello, universe : ";
			Console.Write(s);
			Console.Write(Y);
			Console.WriteLine();
		}
	}
}
```

لنحصل على النتيجة التالية عند التنفيذ:



يمكن للمُبرمج استخدام أحد الأنماط البسيطة التي تظهر في الجدول لتعريف متحولاته. ويؤدي تعريف كل نوع من أنواع المتحولات الظاهرة إلى حجز جزء من الذاكرة يُقدر بالبايت ويتعلق بالنمط المُستَخدَم.

الأنماط المُنمَّرة ENUM

يُعتبر النمط enum من الأنماط البسيطة التي تساعد في تعريف مجموعة من القيم الثابتة بأسمائها الحقيقية:

enum Day { Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday}
By default : Monday=0, Tuesday=1, Wednesday=2, Thursday=3, Friday=4, Saturday=5, Sunday=6

- يمتلك كل عنصر من عناصر النمط enum نمطه الخاص الذي ينتمي إلى أحد الأنماط الصحيحة: byte، أو sbyte، أو short، أو ushort، أو int، أو uint، أو long، أو ulong.
- يكون النمط int هو النمط التلقائي لعناصر النمط enum، ويأخذ العنصر الأول من عناصر النمط enum الرقم 0 ويستمر ترقيم العناصر حتى القيمة n-1 في حال وجود n عنصر.

تقبل لغة C# تعريف أنماط مُنمَّرة مختلفة تمتلك نفس أسماء العناصر مثل:

enum Day { Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday}
enum Weekend { Friday, Saturday, Sunday}

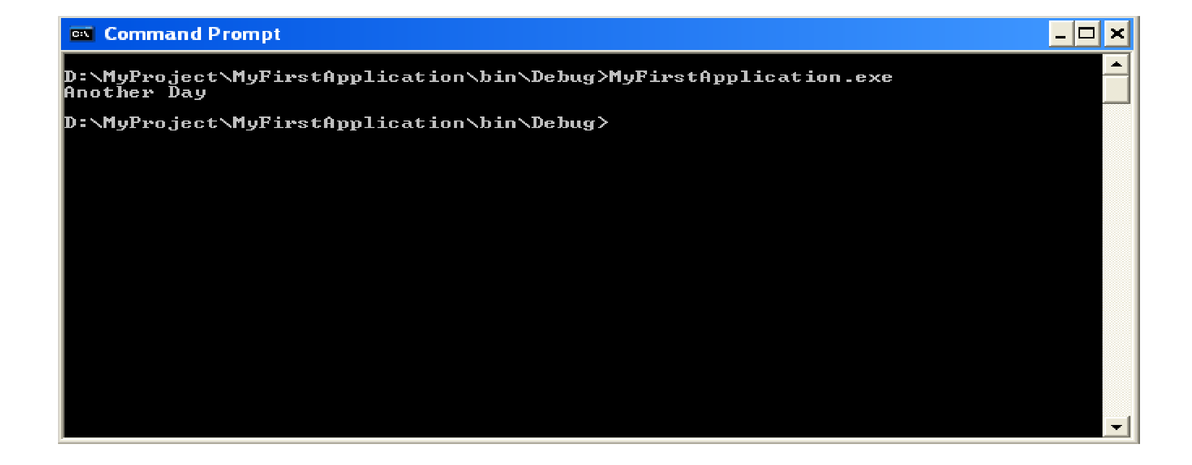
- ولكن يجب الانتباه هنا عند مقارنة عناصر متشابهة من نمطين مختلفين إلى أن النتيجة لاتكون واحد كما هو الحال في البرنامج التالي الذي يعطي النتيجة الظاهرة في الشكل:

```
using System;
namespace MyFirstApplication
{
        class Enum1
        {
                enum Day { Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday}
                enum Weekend { Friday, Saturday, Sunday}

                public static void Main(string[] args)
                {
                        Day ADay ;
                        Weekend  WE;

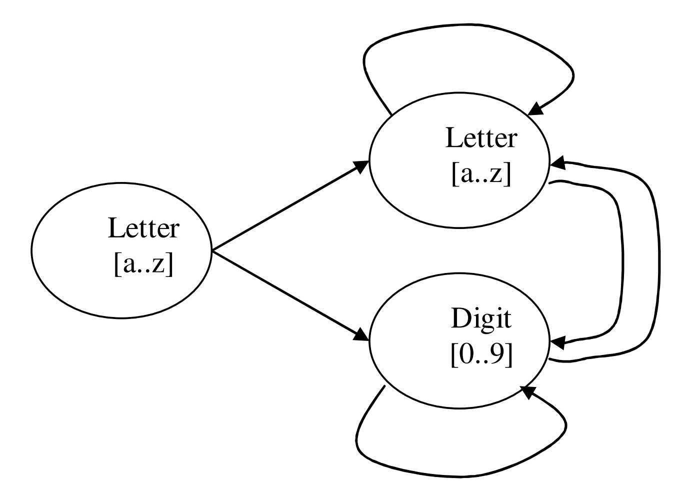
                        ADay= Day.Saturday;
                        WE= Weekend.Saturday;

                        if ( (Day)WE==ADay )
                                System.Console.WriteLine("Same Day");
                        else
                                System.Console.WriteLine("Another Day");
                }
        }
}
```



المتحولات في #C

تبدأ أسماء المتحولات بحرف من الحروف الأبجدية اللاتينية (من a إلى z) تليها اختيارياً سلسلة من الحروف والأرقام وفقاً للمخطط التالي:



بالإضافة لماسبق، يمكن استخدام أحد الحروف التالية: "_"، أو "µ" ، أو الأحرف ذات الإشارات (Characters with Accents) كالأحرف الفرنسية.

مثال:

تعريف متحولات بدون قيم أولية:

```
int GoodMorning ;
int Enumération_fin;
double Value ;
char UnCar ;
bool Test ;
```

تعريف متحولات مع إسناد قيم أولية لها:

```
int GoodMorning=50 ;
int Enumération_fin=10;
double Value=2.3;
char UnCar='K' ;
bool Test=false ;
```

الثوابت في #C

تمتلك لغة #C كلمتين مفتاحيتين للدلالة على المتحولات التي لا يمكن لقيمها أن تتغير: "const" و "readonly"، حيث تُستَخدَم هاتين الكلمتين عند تعريف المتحول أو الثابت قبل أي كلمة مفتاحية أخرى.

يجب على الكلمات المُعرَّفة كـ const أن تأخذ قيم عند تعريفها مبشرةً، ويمكن تعريف عضو من صف أو متحول ضمن إجرائية كـ const فيصبح ثابتاً.

مثال:

```
const int Num=0;   // Accepted
const int Num;     // Error - without Initialization
```

وتسبب عبارة إسناد للثابت Num بحدوث خطأ ناجم عن عدم إمكانية تعديل محتواه:

```
namespace MyFirstApplication
{
      class Test1
      {
            public static void Main()
            {
                  const int Num=0;

                  Num = 10;

                  if ( Num < 0 )
                    System.Console.WriteLine("Negative");
                  else
                    System.Console.WriteLine("Positive");
            }
      }
}
```

أما الكلمات المُعرَّفة كـ readonly فهي تصلح لتكون أعضاء في الصفوف فقط ولايمكن تعريفها ضمن الإجرائيات. ويمكن تركهم دون قيمة إبتدائية عند تعريفهم كأعضاء في الصفوف، بحيث يتم إسناد القيمة الخاصة بهم عند تعريف بنّاء الصفّ فقط (البنّاء هو إجرائية خاصة سيتم تعريف عملها لاحقاً).

مثال:

```
readonly int Num=10;  // Accepted
readonly int Num;  // Accepted
```

العمليات في #C وأفضلياتها

- العمليات الحسابية -

العملية	التوصيف	الأفضلية	مثال
+	إشارة موجبة	1	+a; +b; +1; x=+z;
-	إشارة سالبة	1	-a; -b; -(5+x); y=-(6*u);
++	زيادة بقيمة واحد	1	a++; ++y; v=++c;
--	نقصان بقيمة واحد	1	x--; --z; x=--p;
*	عملية الضرب	2	(a*b); x*y; 6*z; x=a*b;
/	عملية القسمة	2	(a/b); x/5; y=u/y;

%	عملية باقي القسمة	2	x%5; z%a; z=u%t;
+	عملية الجمع	3	a+b; (x+y)+z; t=x+y;
–	عملية الطرح	3	-a-b; x-y-z; t=x-y-z;

تقدم #C كغيرها من لغات البرمجة مجموعة العمليات الحسابية الاعتيادية التي نستعرضها في هذه الشريحة.

العمليات في #C وأفضلياتها

– عمليات المُقارنة –

العملية	التوصيف	الأفضلية	مثال
>	أكبر تماماً	5	a>b; a>(x+y);
>=	أكبر أو يساوي	5	a>=b; a>=(x+y);
<	أصغر تماماً	5	a<b; a<(x+y);
<=	نقصان بقيمة واحد	5	a<=b; a<=(x+y);
==	يساوي	6	a==b; (x+y) == (z+r);
!=	لايساوي	6	a!=b; (x+y) != (z+r);

تقدم #C كغيرها من لغات البرمجة مجموعة عمليات المقارنة المنطقية التي نستعرضها في هذه الشريحة.

العمليات في #C وأفضلياتها

– العمليات المنطقية –

العملية	التوصيف	الأفضلية	مثال
!	نفي	1	! ((a+b) < 6);
&	و	7	((a+b)<6) & ((x+y)>7);
|	أو	9	((a+b)<6) | ((x+y)>7);
&&	"و" مُحسَّنة	10	((a+b)<6) && ((x+y)>7);
||	"أو" مُحسَّنة	11	((a+b)<6) || ((x+y)>7);

تقدم #C كغيرها من لغات البرمجة مجموعة العمليات المنطقية الاعتيادية التي نستعرضها في هذه الشريحة.

العمليات في #C وأفضلياتها
– ملاحظات على العمليات المنطقية –

تُعبِّر عملية ! عن نفي عبارة منطقية وتفترض عدم تحقق الطرف حتى تكون محققة، بحيث يكون جدول الحقيقة للعبارة !((a+b)>8) هو:

a+b > 8	! ((a+b) > 8)
True	False
False	True

تُعبِّر عملية & عن "و" منطقية وتفترض تحقق الطرفين حتى تكون محققة، بحيث يكون جدول الحقيقة للعبارة ((a+b)<6) & ((x+y)>7) مثلاً:

(a+b)<6	(x+y)>7	((a+b)<6) && ((x+y)>7)
True	True	True
True	False	False
False	True	False
False	False	False

تُعبِّر عملية | عن "أو" منطقية وتفترض تحقق أحد الطرفين حتى تكون محققة، بحيث يكون جدول الحقيقة للعبارة ((a+b)<6) | ((x+y)>7) مثلاً:

(a+b)<6	(x+y)>7	((a+b)<6) | ((x+y)>7)
True	True	True
True	False	True
False	True	True
False	False	False

تُعبِّر عملية && عن "و" منطقية أمثلية، كما تُعبِّر عملية || عن "أو" منطقية أمثلية، إذ تمتلك كلتا العمليتان نفس جدول الحقيقة لكل من & و | على الترتيب، إلا أن أهمية هذه العمليات أنه في حالة && مثلاً لاتتم بالضرورة عملية تقييم الطرفين، بل يكفي أن يكون أحد طرفي العبارة (الذي جرى تقييمه أولاً) خطأ حتى يجري اعتبار العبارة خاطئة بكاملها.

تفترض عملية نفي عبارة منطقية عدم تحقق الطرف حتى تكون محققة.

وتفترض عملية "و" منطقية تحقق الطرفين حتى تكون محققة.

في حين تفترض عملية "أو" منطقية تحقق أحد الطرفين حتى تكون محققة.

وتمتلك كلٌ من عملية: "و" الأمثلية وعملية "أو" الأمثلية جدول الحقيقة لكل من عملية "و" العادية و "أو" العادية على الترتيب، إلا أن أهمية هذه العمليات تكمن في أنه، وعلى سبيل المثال، في حالة "و" الأمثلية، لاتتم بالضرورة عملية تقييم الطرفين، بل يكفي أن يكون أحد طرفي العبارة (الذي جرى تقييمه أولاً) خطأً حتى يجري اعتبار العبارة خاطئة بكاملها.

العمليات في #C وأفضلياتها

- أفضليات العمليات -

- يمكن للأقواس أن تحل مشكلة الأفضليات؛
- على سبيل المثال يكون للعبارة ((x+y) > z) && (z<8) التفسير التالي:
  - يجري أولاً حساب x+y ومقارنة النتيجة بقيمة z لتحديد خطأ أو صحة العبارة ((x+y) > z)؛
  - يجري بعدها مقارنة قيمة z بـــ 8 لتحديد خطأ أو صحة العبارة (z<8)؛
  - يجري بعد ذلك التحقق من صحة أو خطأ (z<8) && (x+y) > z)) تبعاً لجدول الحقيقة الخاص بالعملية &&؛
- في حال عدم وجود أقواس يتم تنفيذ العمليات تبعاً للأفضليات (العمليات ذات الأفضلية 1 لها أسبقية على العمليات ذات الأفضلية 2 وهكذا دواليك)؛
- أما في حال تسلسل عمليتين لهما نفس الأفضلية، فتكون الأسبقية للعملية الموجودة على اليسار؛
- على سبيل المثال يكون للعبارة (x+y > z && z<8) التفسير التالي:
  - بما أن عملية "الجمع" تمتلك أسبقية (ذات الأفضلية 3) بالنسبة لعملية المقارنة "أكبر تماماً" (ذات الأفضلية 5) يجري أولاً حساب x+y ومقارنة النتيجة بقيمة z لتحديد خطأ أو صحة العبارة (x+y) > z؛
  - بما أن عملية المقارنة "أصغر تماماً" تمتلك أسبقية (ذات الأفضلية 5) بالنسبة لعملية الـــ "و" المنطقية (ذات الأفضلية 10) يجري أولاً حساب z<8 ومقارنة النتيجة بقيمة z لتحديد خطأ أو صحة العبارة z<8؛
  - يجري بعد ذلك التحقق من صحة أو خطأ (x+y > z && z<8) تبعاً لجدول الحقيقة الخاص بالعملية &&؛

تعليمة القراءة

يمكن لقراءة قيمة متحول ذو نمط بسيط أن نستخدم تعليمة Read أو ReadLine التابعة للصف Console وإسنادها للمتحول المطلوب؛

إلا أن القيمة التي ترجعها Read أو ReadLine تمتلك نمط سلسلة المحارف. فإذا أدخلنا 123 تمت قراءتها من قبل التعليمة على أنها سلسلة المحارف "123".

يؤدي استخدام التعليمة ReadLine دون أخذ الملاحظة الآنفة الذكر بعين الاعتبار إلى حدوث خطأ (عدم توافق الأنماط الناتج عن عند قراءة قيمة ذات نمط محرفي ومحاولة اسنادها لمتحول يعبر عن عدد صحيح) في حال نفذنا البرنامج التالي:

```
using System;
namespace MyFirstApplication
{
        class Test1
        {
                public static void Main()
                {
                        int Num;
                        Console.WriteLine("enter The Requested Value: ");

                        Num=Console.ReadLine();

                        if ( Num < 0 )
                                System.Console.WriteLine("Negative");
                        else
                                System.Console.WriteLine("Positive");
                }
        }
}
```

لذا يتوجب في حال أردنا أن نقرأ متحول من نمط عدد صحيح، أو حقيقي أن نستخدم إجرائيات تحويل خاصة كإجرائية Parse المرتبطة بكل نمط من الأنماط البسيطة والتي تقوم بتحويل سلسلة محارف مثل "123" إلى قيمة هي 123، كما هو الحال في البرنامج التالي:

```
namespace MyFirstApplication
{
        class Test1
        {
                public static void Main()
                {
                        string s;
                        int Num;
                        Console.WriteLine("enter The Requested Value: ");
                        s=Console.ReadLine();
                        Num=Int32.Parse(s);

                        if ( Num < 0 )
                                System.Console.WriteLine("Negative");
                        else
                                System.Console.WriteLine("Positive");
                }
        }
}
```

عموماً، يكون لكل نمط بسيط صف مقابل يمتلك الإجرائية Parse. نورد هذه الصفوف فيمايلي:

النمط	الصف
byte	Byte
sbyte	SByte
short	Int16
ushort	UInt16
int	Int32
uint	UInt32
long	Int64
ulong	UInt64
float	Single
double	Double

تمارين للتجريب

تمرين 1 – نفذ التمرين التالي واستنتج نتيجة التنفيذ:

```
using System;
namespace HelloWorld3s
{
   class Welcome3
   {
      static void Main( string[] args )
      {

         Console.WriteLine( "Welcome\nto\nC#\nProgramming!" );
      }
   }
}
```

تمرين 2– نفذ التمرين التالي واستنتج نتيجة التنفيذ:

```
using System;
using System.Windows.Forms;

namespace WelcomeGUI
{
   class Welcome4
   {
      static void Main( string[] args )
      {
                  MessageBox.Show( "Welcome\nto\nC#\nprogramming!" );
      }
   }
}
```

تمرين 3– نفذ التمرين التالي واستنتج نتيجة التنفيذ:

```
using System;

namespace AdditionProgram
{
   class Addition
   {
      static void Main( string[] args )
      {
         string firstNumber,   // first string entered by user
            secondNumber;  // second string entered by user

         int number1,          // first number to add
            number2,          // second number to add
            sum;              // sum of number1 and number2

         // prompt for and read first number from user as string
         Console.Write( "Please enter the first integer: " );
         firstNumber = Console.ReadLine();

         // read second number from user as string
         Console.Write( "\nPlease enter the second integer: " );
         secondNumber = Console.ReadLine();

         // convert numbers from type string to type int
         number1 = Int32.Parse( firstNumber );
         number2 = Int32.Parse( secondNumber );

         // add numbers
         sum = number1 + number2;

         // display results
         Console.WriteLine( "\nThe sum is {0}.", sum );

      } // end method Main

   } // end class Addition

} // end namespace AdditionProgram
```

تمرين 4– نفذ التمرين التالي واستنتج نتيجة التنفيذ:

```
using System;

namespace ComparisonApplication
{
   class Comparison
   {
      static void Main( string[] args )
      {
         int number1,           // first number to add
            number2;           // second number to add

         // read in first number from user as a string
         Console.Write( "Please enter first integer: " );
         number1 = Int32.Parse( Console.ReadLine() );

         // read in second number from user as a string
         Console.Write( "\nPlease enter second integer: " );
         number2 = Int32.Parse( Console.ReadLine() );

         if ( number1 == number2 )
            Console.WriteLine( number1 + " == " + number2 );

         if ( number1 != number2 )
            Console.WriteLine( number1 + " != " + number2 );

         if ( number1 < number2 )
            Console.WriteLine( number1 + " < " + number2 );

         if ( number1 > number2 )
            Console.WriteLine( number1 + " > " +  number2 );

         if ( number1 <= number2 )
            Console.WriteLine( number1 + " <= " + number2 );

         if ( number1 >= number2 )
            Console.WriteLine( number1 + " >= " + number2 );

      } // end method Main

   } // end class Comparison

} // end namespace ComparisonApplication
```

القسم السابع والثامن والتاسع

التعليمات في لغة #C

الكلمات المفتاحية:

عبارة شرطية، عبارة تكرار، عبارة وصل.

ملخص:

نتعرف في هذا القسم على التعليمات الأساسية في لغة البرمجة #C كالعبارة الشرطية، والعبارة الحلقية، وغيرها.

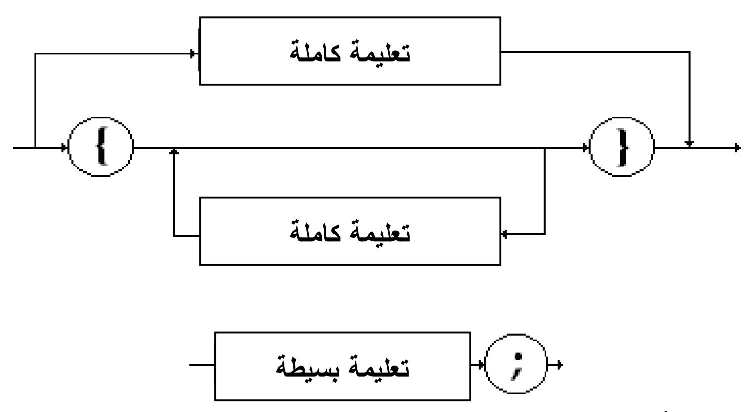
أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- مفهوم كتلة التعليمات؛
- العبارة الشرطية وغموضها واختصارها؛
- عبارات التكرار؛
- عبارة switch … case.

قواعد عامة

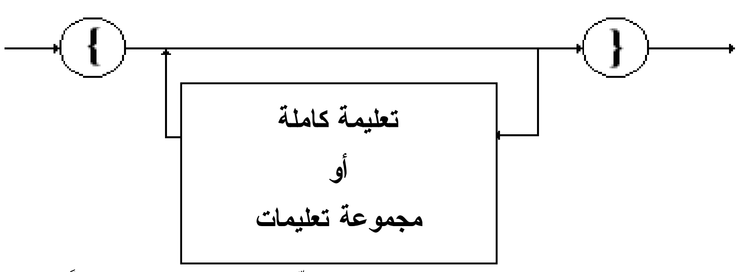
- تم الأخذ بقسم كبير من المعيار ANSI الخاص بلغة C في لغة C#؛
- يمكن أن نكتب تعليمة كاملة محتواة بين قوسين { } أو كتبها دون أقواس؛
- تنتهي أي تعليمة بسيطة (كتعليمة الإسناد) بفاصلة منقوطة؛



- يمكن استخدام تعليقات سطرية تبدأ بالإشارة .... //؛
- يمكن أن استخدام تعليقات نصيّة في برنامج مكتوب بلغة C# بإحاطتها بـــ /* ... */.

كتل التعليمات وتعريف مدى المتحول

نُعرِّف كتلة تعليمات كمايلي:



يمكن لكتلة التعليمات أن تحتوي كتلة تعليمات أخرى على أن تكون الكتل معلّبة ببعضها البعض تماماً، إذ لايمكن أن تتقاطع كتلتي تعليمات جزئياً بأن تكون بداية الثانية بعد بداية الأولى وأن تكون نهاية الثانية بعد نهاية الأولى مثلاً.

مثال:

```
int a, b = 12;
  { int x , y = 8 ;
   { int z =12;
        x = z ;
        a = x + 1 ;
      { int u = 1 ;
        y = u - b ;
      }
   }
  }
```

نُعرِّف مدى المتحول بالكتلة التي يكون المتحول فيها مُعرفاً، ويكون المتحول مُعرّفاً في الكتلة التي تم تعريفه فيها وفي جميع الكتل المحتواة في كتلته، ولكنه لايكون مُعرّفاً في الكتل التي تحوي كتلته أو الكتل الموازية لكتلته.

مثال:

ليكن لدينا البرنامج التالي:

```
//Block0
int a, b = 12;

//Block1
{
      int x , y = 8 ;
}

// Block2
{
      int z =12;
      a = b + z;
      x = z ; //error
      a = x + 1 ; //error
}
//Block3
{
      int u = 1 ;
      a = b - u;
      y = u - b ; //error
}
```

تظهر الأخطاء نتيجة عدم وجود أي تعريف لكل من x و y في الكتل التي تظهر بها، في حين لا توجد أي مشكلة في استخدام المتحولات a و b في هذه الكتل.

تعليمة الإسناد

الإسناد البسيط:

يكون رمز الإسناد هو "=" حيث نكتب:

x=y

يجب في هذه الحالة أن تكون x هي مُعرِّف متحول.

يمكن استخدام الإسناد ضمن العبارات الحسابية والمنطقية، وعلى شكل إسناد متعدد.

مثال:

```
int a , b = 56, c, d ;
a = (b = 12)+8;  // New value of b
a = b = c = d =8; // Multiple Assignment
```

في الحالة الأولى تأخذ b قيمة أولية عند تعريفها، أما في الحالة الثانية فتأخذ b قيمة جديدة عند استخدامها داخل العبارة، وتأخذ b قيمة جديدة هي 8 في الحالة الثالثة ولكن ضمن عملية إسناد متعددة.

وتكون قيم المتحولات في الحالات الثلاث بعد انتهاء عمليات الإسناد:

`int a , b = 56, c, d ;`	`a=???, b=56, c=???, d=???`
`a = (b = 12)+8;`	`a=20, b=8`
`a = b = c = d =8;`	`a=8, b=8, c=8, d=8`

الإسناد المزود بعملية:

لتكن **op** إحد العمليات التالية: {+, -, *, /, &, |}.

من الممكن أن نستخدم إسناد مزود بعملية من العمليات السابقة له الشكل:

x op= y;

بشكل مكافئ للإسناد البسيط التالي:

x = x op y;

مثال:

```
int a , b = 56 ;
a = -8 ;
a += b ;  // a = a + b
b *= 3 ;  // b = b * 3
```

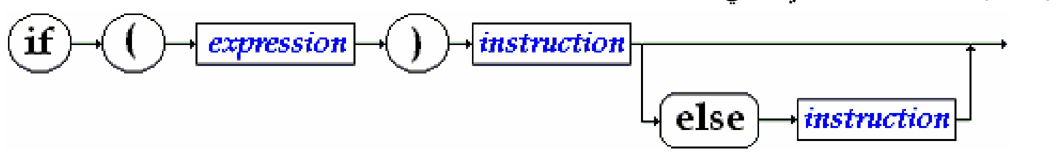
تكون قيم المتحولات في الحالات الثلاث بعد انتهاء عمليات الإسناد:

`int a , b = 56 ;`	`a=???, b=56`
`a = -8 ;`	`a=-8`
`a += b ;`	`a=46`
`b *= 3 ;`	`b=168`

يمكن استخدام الإسناد ضمن العبارات الحسابية والمنطقية، وعلى شكل إسناد متعدد. كما يمكن استخدام إسناد مزود بعملية من العمليات الحسابية أو المنطقية بشكل مكافئ للإسناد البسيط.

العبارة الشرطية

تأخذ العبارة الشرطية الشكل القواعدي التالي:



بحيث يكون لأي شرط أحد شكلين:

```
if ( Expression ) Instructions Bloc ;

                    أو

if ( Expression ) Instructions Bloc ; else Instructions Bloc ;
```

مثال:

```
int a=100, b=0, c;

if ( b == 0 )
{
        c = 1;
        System.Console.WriteLine("First Condition: c = " + c);
}
else
{
        c = a / b;
        System.Console.WriteLine("First Condition: c = " + c);
}

if ((c = a*b) != 0)
        c += b;
else
c = a;

System.Console.WriteLine("Second Condition: c = " + c);
```

وتكون النتيجة:

```
C:\MyProject\MyFirstApplication\bin\Debug\MyFirstApplication.exe
First Condition: c = 1
Second Condition: c = 100
```

غموض العبارة الشرطية

يمكن أن تظهر العبارة الشرطية بشكل غامض كما هو الحال في المثال التالي:

```
if ( x>0 )
if ( y+z>10 )
x=x+5 ;
else
x=x-5;
```

في هذه الحالة يظهر الغموض في تحديد تبعية عبارة else لعبارة if.

بشكل عام تكون عبارة else تابعة لعبارة if الأقرب إلا إذا حددت الأقواس عكس ذلك. ففي الحالة السابقة تكون التبعية كمايلي:

```
if ( x>0 )
        if ( y+z>10 )
                x=x+5 ;
        else
                x=x-5;
```

طبعاً، يمكن للأقواس في حال استخدامها أن تحلّ الغموض وفقاً لتوزعها. ففي الحالتين التاليتين تُزيل الأقواس الغموض تماماً وتحدد تبعية عبارة else لعبارة if:

```
if ( x>0 )
{
        if ( y+z>10 )
                x=x+5 ;
        else
                x=x-5;
}
```

```
if ( x>0 )
{
        if ( y+z>10 )
                x=x+5 ;
}
else
        x=x-5;
```

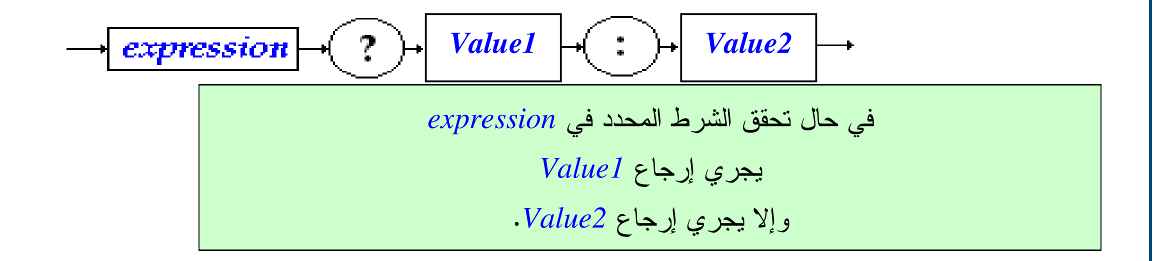
يمكن أن تظهر العبارة الشرطية بشكل غامض. حيث يظهر الغموض في تحديد تبعية عبارة else لعبارة if.

بشكل عام تكون عبارة else تابعة لعبارة if الأقرب إلا إذا حددت الأقواس عكس ذلك.

طبعاً، يمكن للأقواس في حال استخدامها أن تحلّ الغموض وفقاً لتوزعها.

اختصار العبارة الشرطية

يمكن في بعض الحالات عندما يكون الهدف من العبارة الشرطية تنفيذ عملية إسناد أن نبني عبارة شرطية مختصرة لها الشكل القواعدي التالي:



وتكافئ التعليمة السابقة التعليمة:

if (*expression*) *Value1* else *Value2*

مثال:

نقرأ العبارة الشرطية المُختصرة التالية:

```
int a,b,c ;
….
c =   a == 0  ?  b  :  a+1 ;
```

كما يلي:

في حال تحققت العبارة (a==0) يجري إرجاع القيمة الأولى (b) وإسنادها إلى (c) وإلا فإن القيمة الثانية (a+1) هي التي يجري إرجاعها وإسنادها إلى (c).

يمكن في بعض الحالات عندما يكون الهدف من العبارة الشرطية تنفيذ عملية إسناد أن نبني عبارة شرطية مختصرة.

عبارات التكرار: while، do، for

1. يكون لعبارة التكرار الحلقية while الشكل القواعدي التالي:



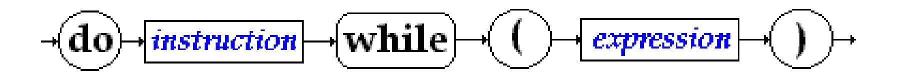
- يجري أولا اختبار الشرط *expression* الذي يعيد قيمة منطقية صح أو خطأ
  - فإذا كانت صح يجري تنفيذ *instruction*؛
  - وإلا يجري الخروج دون تنفيذ *instruction*.
- بعد كل تنفيذ للتعليمات *instruction*، يجري اختبار الشرط *expression* للتأكد من أن قيمته المنطقية مازالت صح:
  - فإذا كانت صح نستمر في تكرار *instruction* مرة أخرى؛
  - وإلا يجري التوقف عن تكرار *instruction*.

مثال:

```
int i=0;
int x=0;

while (i<=10)
{
      x=x+10;
      i++;
}
```

2. ويكون لعبارة التكرار الحلقية do الشكل القواعدي التالي:



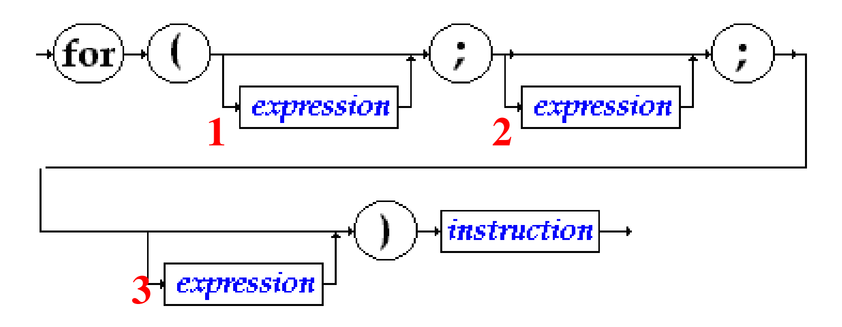
- يجري أولا تنفيذ التعليمات *instruction*
- يجري بعدها اختبار الشرط *expression*:
  - فإذا كان صحيحاً نستمر في تنفيذ *instruction*.
  - وإلا يجري التوقف.
- بعد كل تنفيذ للتعليمات *instruction*، يجري اختبار الشرط *expression* للتأكد من أن قيمته المنطقية مازالت صح:
  - فإذا كانت صح نستمر في تكرار *instruction* مرة أخرى؛
  - وإلا يجري التوقف عن تكرار *instruction*.

مثال:

```
int j=0;
int y=0;

do
{
      y=y+10;
      j++;
} while (j<=10);
```

3. ويكون لعبارة التكرار الحلقية for الشكل القواعدي التالي:



- يجري أولا تنفيذ عبارة الإعداد *expression* المُشار إليها بالرقم 1؛
- يجري بعدها اختبار الشرط الموجود في العبارة الشرطية *expression* المُشار إليها بالرقم 2:
  - فإذا كان الشرط صحيحاً نبدأ في تنفيذ *instruction*.
  - وإلا يجري التوقف.
- بعد كل تنفيذ للتعليمات *instruction*، يجري تنفيذ التعليمات الموجودة في عبارة التعديل *expression* المُشار إليها بالرقم 3 وإعادة اختبار الشرط الموجود في العبارة الشرطية *expression* المُشار إليها بالرقم 2 للتأكد من أن قيمتها المنطقية مازالت صح:
  - فإذا كانت صح نستمر في تكرار *instruction* مرة أخرى؛
  - وإلا يجري التوقف عن تكرار *instruction*.

مثال:

```
int z=0;

for (k=0; k<=10; k++)
      z=z+10;
```

مثال:

تكون العبارات الثلاث متكافئة في البرنامج التالي:

```
int x=0;
int y=0;
int z=0;

int i=0;
while (i<=10)
{
      x=x+10;
      i++;
}
Console.WriteLine("x = " + x);

i=0;
do
{
      y=y+10;
      i++;
} while (i<=10);
Console.WriteLine("y = " + y);

for (i=0; i<=10; i++)
      z=z+10;
Console.WriteLine("z = " + z);
```

وتكون النتيجة:

```
C:\MyProject\MyFirstApplication\bin\Debug\MyFirstApplication.exe
x = 110
y = 110
z = 110
```

في حين لا يكون هناك تكافؤ في الحالة التالية:

```
int x=0;
int y=0;
int z=0;

int i=11;
while (i<=10)
{
      x=x+10;
      i++;
}
Console.WriteLine("x = " + x);

i=11;
do
{
      y=y+10;
      i++;
} while (i<=10);
Console.WriteLine("y = " + y);

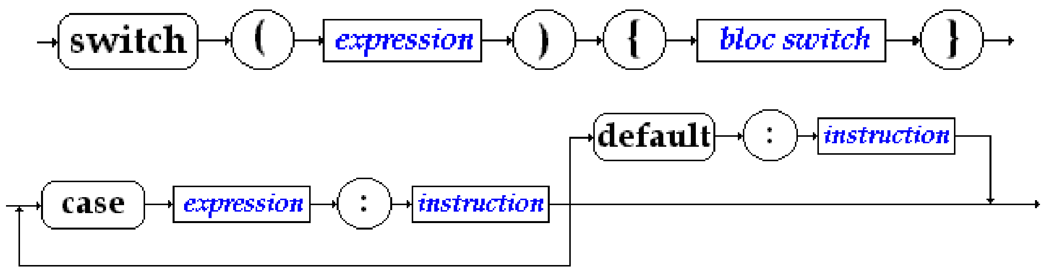
for (i=11; i<=10; i++)
      z=z+10;
Console.WriteLine("z = " + z);
```

وتكون النتيجة:

```
C:\MyProject\MyFirstApplication\bin\Debug\MyFirstApplication.exe
x = 0
y = 10
z = 0
```

عبارة الوصل: switch ... case

يكون لعبارة switch ... case الشكل القواعدي التالي:



نقرأ العبارة كمايلي:

- بعد التعرُّف على عبارة ***expression*** ندخل إلى ***bloc switch***:
  - نطابق بين العبارة ***expression*** الخارجية التي تعرفنا عليها عند مدخل ***bloc switch*** مع العبارات ***expression*** الداخلية المُحددة داخله؛
  - عند حدوث تطابق يجري تنفيذ التعليمات ***instruction*** المقابلة؛
  - في حال وجود عدة حالات تطابق بين عبارة ***expression*** الخارجية مع عبارات ***expression*** في الداخل، يجري تنفيذ التعليمات المرتبطة بعبارات ***expression*** الداخلية المطابقة حسب تسلسل ظهورها؛
  - إذا أردنا أن يقتصر التنفيذ على العبارة المطابقة الأولى فقط، توجب إضافة تعليمة ***break*** إلى نهاية التعليمات المرتبطة بكل عبارة ***expression*** داخلية؛
  - في حال عدم وجود أي تطابق مع العبارات ***expression*** الداخلية، يجري تنفيذ التعليمات ***instruction*** المرتبطة بالعبارة ***default***.

مثال:

```
int  x = 10;
switch (x+1)
{
      case 11 : Console.WriteLine(">> case 11");
            break;

      case 12 : Console.WriteLine(">> case 12");
            break;

      default : Console.WriteLine(">> default");
            break;
}
```

تكون النتيجة هي تطابق العبارة (x+1) مع الحالة الأولى أي (case 11).

تمرين

اكتب صف Average1 بلغة C#، يساعد في حساب المتوسط الحسابي لعشرة أرقام صحيحة يجري طلبها من البرنامج وإدخالها من قبل المُستَخدِم.

الحل:

```
using System;
namespace Average
{
   class Average1
   {
      static void Main( string[] args )
      {
         int total,          // sum of grades
            gradeCounter,    // number of grades entered
            gradeValue,      // grade value
            average;         // average of all grades

         // initialization phase
         total = 0;           // clear total
         gradeCounter = 1;    // prepare to loop

         // processing phase
         while ( gradeCounter <= 10 )  // loop 10 times
         {
            // prompt for input and read grade from user
            Console.Write( "Enter integer grade: " );
            // read input and convert to integer
            gradeValue = Int32.Parse( Console.ReadLine() );
            // add gradeValue to total
            total = total + gradeValue;
            // add 1 to gradeCounter
            gradeCounter = gradeCounter + 1;
         }

         // termination phase
         average = total / 10;  // integer division
         // display average of exam grades
         Console.WriteLine( "\nClass average is {0}", average );

      } // end Main

   } // end class Average1
}
```

تمرين

اكتب صف Average2 بلغة C#، يساعد في حساب المتوسط الحسابي لعدد من الأرقام صحيحة يجري طلبها من البرنامج وإدخالها من قبل المُستَخدِم وبحيث يتوقف طلب الأعداد عند إدخال المستخدم للقيمة 1-.

الحلّ:

```
using System;
namespace Average
{
   class Average2
   {
      static void Main( string[] args )
      {
         int total,           // sum of grades
            gradeCounter,    // number of grades entered
            gradeValue;      // grade value
         double average;      // average of all grades

         // initialization phase
         total = 0;           // clear total
         gradeCounter = 0;    // prepare to loop

         // processing phase
         // prompt for input and convert to integer
         Console.Write( "Enter Integer Grade, -1 to Quit: " );
         gradeValue = Int32.Parse( Console.ReadLine() );

         // loop until a -1 is entered by user
         while ( gradeValue != -1 )
         {
            // add gradeValue to total
            total = total + gradeValue;

            // add 1 to gradeCounter
            gradeCounter = gradeCounter + 1;

            // prompt for input and read grade from user
            // convert grade from string to integer
            Console.Write( "Enter Integer Grade, -1 to Quit: " );
            gradeValue = Int32.Parse( Console.ReadLine() );
         } // end while

         // termination phase
         if ( gradeCounter != 0 )
         {
            average = ( double ) total / gradeCounter;
            // display average of exam grades
            Console.WriteLine( "\nClass average is {0}", average );
         }
         else
         {
            Console.WriteLine( "No grades were entered." );
         }

      } // end method Main
   } // end class Average2
}
```

تمرين

بفرض أن لديك مجموعة من 10 طلاب، اكتب صف Analysis بلغة C# يستعرض أرقام الطلاب من 1 إلى 10 رقماً رقماً، بحيث يقوم المستخدم من أجل كل طالب بإدخال رقم 1 في حال كان الطالب ناجحاً، وإدخال رقم 2 في حال كان الطالب راسباً، وبحيث يعطي البرنامج في النهاية عدد الناجحين وعدد الراسبين والنسبة المئوية للنجاح.

الحل:

```
using System;

namespace Passes_and_Failures
{
   class Analysis
   {
      static void Main( string[] args )
      {
         int passes = 0,             // number of passes
            failures = 0,           // number of failures
            student = 1,            // student counter
            result;                 // one exam result

         // process 10 students; counter-controlled loop
         while ( student <= 10 )
         {
            Console.WriteLine( "Enter result (1=pass, 2=fail)" );
            result = Int32.Parse( Console.ReadLine() );

            if ( result == 1 )
               passes = passes + 1;

            else
               failures = failures + 1;

            student = student + 1;
         }

         // termination phase
         Console.WriteLine();
         Console.WriteLine( "Passed: " + passes );
         Console.WriteLine( "Failed: " + failures );

         Console.WriteLine( "Raise Percentage\n" + passes * 10 );

      } // end of method Main

   } // end of class Analysis

}
```

تمرين

جرّب البرنامج التالي وأبد ملاحظاتك على عمليات الزيادة بقيمة 1.

```
using System;

namespace Increment
{
   class Increment
   {
      [STAThread]
      static void Main(string[] args)
      {
         int c;

         c = 5;
         Console.WriteLine( c );   // print 5
         Console.WriteLine( c++ ); // print 5 then postincrement
         Console.WriteLine( c );   // print 6

         Console.WriteLine();      // skip a line

         c = 5;
         Console.WriteLine( c );   // print 5
         Console.WriteLine( ++c ); // preincrement then print 6
         Console.WriteLine( c );   // print 6s

      } // end of method Main

   } // end of class Increment
}
```

تمرين

اكتب برنامج بلغة C# لحساب مجموع الأعداد الزوجية المحصورة بين 0 و 255.

الحلّ:

```
using System;
using System.Windows.Forms;

namespace MySum
{
   class Sum
   {
      static void Main( string[] args )
      {
         int sum = 0;

         for ( int number = 2; number <= 100; number += 2 )
            sum += number;

         MessageBox.Show( "The sum is " + sum,
            "Sum Even Integers from 2 to 100",
            MessageBoxButtons.OK,
            MessageBoxIcon.Information );

      } // end of method Main

   } // end of class Sum
}
```

تمرين

بفرض أن لديك مجموعة من 10 طلاب، اكتب صف Eval بلغة #C يستعرض أرقام الطلاب من 1 إلى 10 رقماً رقماً، بحيث يقوم المستخدم من أجل كل طالب بإدخال تقييمه الذي يمكن أن يكون إما A أو B أو C أو D، وبحيث يعطي البرنامج في النهاية عدد التقييمات من مرتبة A، و عدد التقييمات من مرتبة B، عدد التقييمات من مرتبة C، عدد التقييمات من مرتبة D، مع العلم أنه بإمكان المُستخدِم إدخال الحرف A أو a للدلالة على التقييم A، وإدخال الحرف b أو B للدلالة على التقييم B وهكذا دواليك.



```
using System;

namespace Evaluation
{

   class Eval
   {
      static void Main( string[] args )
      {
         char grade;       // one grade
         int aCount = 0,   // number of As
            bCount = 0,    // number of Bs
            cCount = 0,    // number of Cs
            dCount = 0,    // number of Ds
            fCount = 0;    // number of Fs

         for ( int i = 1; i <= 10; i++ )
         {
            Console.Write( "Enter a letter grade: " );

            grade = Char.Parse( Console.ReadLine() );

            switch ( grade )
            {
               case 'A':   // grade is uppercase A
               case 'a':   // or lowercase a
                  ++aCount;
                  break;

               case 'B':   // grade is uppercase B
               case 'b':   // or lowercase b
                  ++bCount;
                  break;

               case 'C':   // grade is uppercase C
               case 'c':   // or lowercase c
                  ++cCount;
                  break;

               case 'D':   // grade is uppercase D
               case 'd':   // or lowercase d
                  ++dCount;
                  break;

               case 'F':   // grade is uppercase F
               case 'f':   // or lowercase f
                  ++fCount;
                  break;

               default:    // processes all other characters
                  Console.WriteLine(
                     "Incorrect letter grade entered." +
                     "\nEnter a new grade" );
                  i--;
                  break;

            } // end switch

         } // end for

         Console.WriteLine(
            "\nTotals for each letter grade are:\nA: {0}" +
            "\nB: {1}\nC: {2}\nD: {3}\nF: {4}", aCount, bCount,
            cCount, dCount, fCount );

      } // end method Main

   } // end class Eval

} // end namespace Evaluation
```

تمرين

نفذ البرنامج التالي وأعط نتيجته:

```
using System;
namespace LogicalOperators
{
   class LogicalOperators
   {
      [STAThread]
      static void Main(string[] args)
      {
         // testing the conditional AND operator (&&)
         Console.WriteLine( "Conditional AND (&&)" +
            "\nfalse && false: " + ( false && false ) +
            "\nfalse && true:  " + ( false && true ) +
            "\ntrue && false:  " + ( true && false ) +
            "\ntrue && true:   " + ( true && true ) );

         // testing the conditional OR operator (||)
         Console.WriteLine( "\n\nConditional OR (||)" +
            "\nfalse || false: " + ( false || false ) +
            "\nfalse || true:  " + ( false || true ) +
            "\ntrue || false:  " + ( true || false ) +
            "\ntrue || true:   " + ( true || true ) );

         // testing the logical AND operator (&)
         Console.WriteLine( "\n\nLogical AND (&)" +
            "\nfalse & false: " + ( false & false ) +
            "\nfalse & true:  " + ( false & true ) +
            "\ntrue & false:  " + ( true & false ) +
            "\ntrue & true:   " + ( true & true ) );

         // testing the logical OR operator (|)
         Console.WriteLine( "\n\nLogical OR (|)" +
            "\nfalse | false: " + ( false | false ) +
            "\nfalse | true:  " + ( false | true ) +
            "\ntrue | false:  " + ( true | false ) +
            "\ntrue | true:   " + ( true | true ) );

         // testing the logical exclusive OR operator (^)
         Console.WriteLine( "\n\nLogical exclusive OR (^)" +
            "\nfalse ^ false: " + ( false ^ false ) +
            "\nfalse ^ true:  " + ( false ^ true ) +
            "\ntrue ^ false:  " + ( true ^ false ) +
            "\ntrue ^ true:   " + ( true ^ true ) );

         // testing the logical NOT operator (!)
         Console.WriteLine( "\n\nLogical NOT (!)" +
            "\n!false: " + ( !false ) +
            "\n!true:  " + ( !true ) );

      }
   }
}
```

الحلّ:

```
C:\Data\Data\Formations\Cshrp C
Conditional AND (&&)
false && false: False
false && true:  False
true && false:  False
true && true:   True

Conditional OR (||)
false || false: False
false || true:  True
true || false:  True
true || true:   True

Logical AND (&)
false & false: False
false & true:  False
true & false:  False
true & true:   True

Logical OR (|)
false | false: False
false | true:  True
true | false:  True
true | true:   True

Logical exclusive OR (^)
false ^ false: False
false ^ true:  True
true ^ false:  True
true ^ true:   False

Logical NOT (!)
!false: True
!true:  False
```

القسم العاشر

المفاهيم الأساسية للتقانة غرضية التوجه

الكلمات المفتاحية:

غرضية التوجه، الغرض، الصف، مميز هوية الغرض، الطرائق، الوراثة، الكبسلة، هرمية الصفوف، تعددية الأشكال، إعادة تعريف الطرائق.

ملخص:

يُركِّز هذا الفصل على المفاهيم الأساسية للتقانة غرضية التوجه.

أهداف تعليمية:

يهدف هذا الفصل إلى:

- التعرف على الأغراض واصفاتها وطرائقها.
- تعريف الصفوف والعلاقات بينها.
- الوراثة الأحادية والمتعددة.
- إعادة التعريف وتعددية الأشكال.
- العلاقات بين الأغراض.

مقدمة

- **المنهجية غرضية التوجه:** عبارة عن منهجية للتطوير والنمذجة مبنية على مفاهيم وأغراض.
- **البرمجة غرضية التوجه:** هي طريقة بديلة عن البرمجة التقليدية، تعتمد على أغراض، حيث يحتوي كل غرض على معطيات وطرائق للتعامل معه.
- **الغرض:** يُمثِّل كل غرض كيان من العالم الحقيقي مُميَّز الهوية مع واصفات مميزة وإمكانية العمل بنفسه والتفاعل مع الأغراض الأخرى.
- **مثال:**
  - الغرض: شخص
  - واصفاته: رقم الضمان الاجتماعي، الاسم الأول، الاسم الثاني، تاريخ الميلاد، العنوان.

تُعتبر البرمجة غرضية التوجه طريقة بديلة عن البرمجة التقليدية، وتعتمد هذه البرمجة على استخدام مفهوم الغرض، حيث يُمثِّل كل غرض كيان من العالم الحقيقي مُميَّز الهوية مع واصفات مميزة، وإمكانية العمل بنفسه والتفاعل مع الأغراض الأخرى.

يتم تعريف الغرض من خلال مُميِّز الهوية الخاص به، حيث يُسنَد للغرض لحظة بنائه ولايمكن تغييره أبداً، ويُحذَف لحظة حذف الغرض.

يجري تحديد كل غرض من خلال مجموعة من الواصفات، وتتميز النسخ المختلفة من الغرض عن طريق القيم المختلفة للواصفات، حيث تملك كل نسخة قيم مختلفة لهذه الواصفات.

أساسيات التقانة غرضية التوجه

- يتألف العالم من حولنا من أغراض يكون كل منها في حالة محددة تحددها القيم الحالية لصفات الغرض.
- لكل غرض من الأغراض الحقيقية هوية (identity)، وهي خاصة ثابتة نميز من خلالها بين غرض وآخر.

قد يكون إجراء محاكاة مع أغراض حسية من الحياة الواقعية طريقة جيدة لشرح وتوضيح المفاهيم غرضية التوجه، إذ يتألف العالم من حولنا من أغراض يكون كل منها في حالة محددة تحددها القيم الحالية لصفات الغرض.

فعلى سبيل المثال يتواجد فنجان القهوة على مكتبي في الحالة "مملوء" لأنه مصمم بحيث يستوعب السوائل ومازالت القهوة موجودة فيه، وعندما لا تبقى هناك قهوة في الفنجان يصبح في الحالة "فارغ" وإذا سقط على الأرض وتحطم سيصبح في الحالة "مكسور".
إلا أن فنجان القهوة كائن سلبي، فهو لا يتميز بسلوك خاص (behavior)، بالمقابل لا يمكن قول الأمر نفسه بالنسبة لكلب أو شجرة، فالكلب

ينبح، والشجرة تنمو، وللأغراض الحقيقية عادة سلوك.

لكل غرض من الأغراض الحقيقية هوية (identity)، وهي خاصة ثابتة نميز بواسطتها بين غرض وآخر، فإذا كان على مكتبي فنجانا قهوة من المجموعة نفسها يمكنني القول إن الفنجانين متساويان لكنهما غير متطابقين، فهما متساويان لأن لهما قيم الخصائص نفسها و لهما الشكل والحجم نفسه، وكلاهما أسود مثلاً، لكنهما ليسا متطابقين في اللغة غرضية التوجه، لأن هناك اثنان يمكنني أن أستخدم أياً منهما وفقا لرغبتي.

الغرض

- يتألف النظام غرضي التوجه من مجموعة أغراض متعاونة، فكل شيء في النظام غرضي التوجه هو عبارة عن غرض.
- يُعبَّر عن الغرض في لغة غرضية التوجه كمستطيل من خلال الصف الذي ينتمي إليه؛
- يجري تدوين العمليات الخاصة بغرض ضمن الصف الذي ينتمي إليه الغرض لأنها مشتركة بين جميع الأغراض.

الغرض هو مثل من "شيء"، فقد يكون من أمثلة عديدة للشيء نفسه، ففنجان القهوة الموجود لدي هو مثل من مجموعة الفناجين الموجودة.
يتألف النظام غرضي التوجه من مجموعة أغراض متعاونة، فكل شيء في النظام غرضي التوجه هو عبارة عن غرض.
من المهم أن نشير هنا إلى أن تدوين الغرض لا يحوي جزءاً للعمليات (Operations) التي يمكن أن ينفذها الغرض، ويعود هذا إلى كون العمليات متطابقة في كل الأمثال ولن يكون تكرار ذكرها في كل مثل مناسباً. لذلك يجري تخزين العمليات على مستوى الصف.

واصفات الغرض

- يمكن أن تأخذ الواصفة قيمة واحدة أو عدة قيم.
- يمكن أن تُشير واصفات الغرض إلى غرض أو عدة أغراض أخرى.
- حالة الغرض: هي عبارة عن مجموعة القيم التي تأخذها واصفاته في لحظة معينة، ويمكن أن تتغير هذه الحالة، في حين يبقى مميز هوية الغرض نفسه.

يمكن أن تُشِير واصفات الغرض إلى غرض أو عدة أغراض أخرى، كما يمكن أن تأخذ قيمة واحدة أو عدة قيم.

يُستخدَم مميز هوية الغرض من أجل الربط بين غرضين، فمثلاً لدينا الغرض (طالب) وواصفة لهذا الغرض (المواد الدراسية) تشير إلى مجموعة من المواد. عندها تحوي الواصفة (المواد الدراسية) على مميز هوية غرض يحوي على قائمة من أغراض المواد، وبذلك يتم الربط بين الغرضين.

طرائق الغرض

- الطريقة: هي عبارة عن رماز يُستخدَم لتنفيذ عملية معينة على واصفات الغرض، ولكل طريقة اسم ويمكن أن تملك مجموعة معاملات.

- كل العمليات التي يمكن تنفيذها على الغرض يجب أن تتحقق من خلال طرائق.
- الكبسلة: هي إمكانية اخفاء البنية الداخلية للغرض (الواصفات والطرائق) عن الأغراض الأخرى.

الطريقة: هي عبارة عن رماز يستخدم لتنفيذ عملية معينة على واصفات الغرض، حيث يتم تحقيق كل العمليات التي يمكن تنفيذها على الغرض من خلال طرائق. فالطريقة هي التي تحمي واصفات الغرض من الوصول إليها.

يجري طلب تنفيذ طريقة معينة من غرض معين من خلال إرسال رسالة إلى الغرض تحوي اسم الطريقة والمعاملات المطلوبة لتنفيذها، حيث يقوم الغرض المعني بتنفيذ الطريقة وإعادة النتيجة في حال وجودها.

من الواضح أن أي غرض لا يمكن أن يصل للبنية الداخلية لغرض آخر، إنما يتم التخاطب بينهما عن طريق رسائل تتضمن طلب تنفيذ طرائق معينة. وهذا ما يدعى بمفهوم الكبسلة أي إمكانية إخفاء البنية الداخلية للغرض (الواصفات والطرائق) عن الأغراض الأخرى.

تمارين

1- اقترح غرضاً لتمثيل الوقت يحوي على مجموعة من الواصفات التي تعبر عن الوقت على نحو دقيق وعلى مجموعة من الطرائق التي تتعامل مع هذه الواصفات وتسمح بقراءتها أو تعديلها.

الحل:

Class Time:
- Class Attributes:
  - Hours;
  - Minutes;
  - Seconds;
- Class Methods:
  - Set_Time(H, M, S);
  - Get_Time( );

2- اقترح غرضاً لتمثيل شخص (ذاتية شخص) واقترح مجموعة من الطرائق للتعامل مع واصفاته.

الحل:

Class Person:
- Class Attributes:
  - First_Name;
  - Second_Name;
  - National_ID;
  - Address;
  - Date_Of_Birth;
- Class Methods:

- Set_FullName(FN, SN);
- Get_FullName( );
- Set_NationalID( );
- Get_NationalID( );
- Set_Address(A);
- Get_Address( );
- Set_Date_Of_Birth(DOB)
- Get_Date_Of_Birth( );

الصفوف

- الصف هو عبارة عن مجموعة من الأغراض المتشابهة مع بنية متشابهة (واصفات) وسلوك متشابه (طرائق).
- الصف هو الواصف لمجموعة من الأغراض لها الصفات نفسها والعمليات نفسها، وهو يلعب بذلك دور قالب لإنشاء الأغراض.
- يدعى كل غرض من الصف بـــ(نسخة الصف)، حيث تملك كل نسخة مميز هوية فريد، وتستطيع أن تستدعي الطرائق المعرّفة ضمن الصف.
- هنالك نوعان لطرائق الصف، طرائق عامة يمكن طلبها من أغراض أخرى، وطرائق خاصة لا يمكن أبداً طلب تنفيذها من أغراض أخرى.

يتم تجميع الأغراض التي تمتلك معطيات متشابهة في صفوف، فالصف هو عبارة عن مجموعة من الأغراض المتشابهة مع بنية متشابهة (واصفات) وسلوك متشابه (طرائق). فالصف هو الواصف لمجموعة من الأغراض لها الصفات نفسها والعمليات نفسها، وهو يلعب بذلك دور قالب لإنشاء الأغراض.

يحتوي كل غرض يجري إنشاؤه بهذا القالب، على قيم الصفات التي تتوافق مع الأنماط المعرفة ضمن تعريف الصف، كما يستطيع أي غرض أن يستدعي العمليات المعرفة في صفه.

يحتوي الصف على وصف لبنية المعطيات وتفصيل تحقيق الطرائق للأغراض في ذلك الصف.
يُدعى كل غرض من الصف بـــ(نسخة الصف)، وكل نسخة تملك مميز هوية فريد، كما أنها تعرف الصف الذي تنتمي إليه.

هنالك نوعان لطرائق الصف، طرائق عامة وطرائق خاصة، حيث يمكن لأغراض أخرى أن تطلب تنفيذ الطرائق العامة فقط، بينما لا يمكنها أبداً طلب تنفيذ الطرائق الخاصة.

مثال 1

- نستعرض في هذا المثال الصف Point الذي يحتوي على واصفتين X و Y تمثلان إحداثيات النقطة، والطريقة Distance لحساب البعد بين نقطتين، والطريقة Equals لمقارنة نقطتين (متساويتين أم لا).

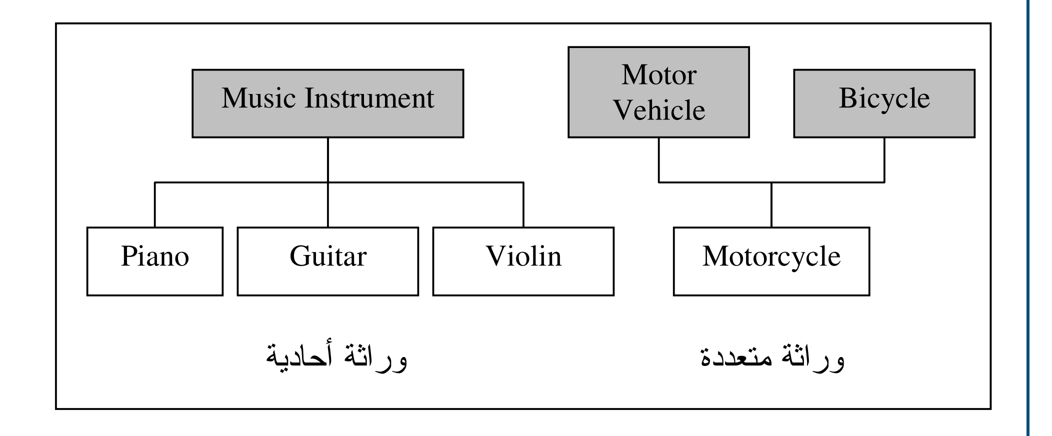
```
CLASS Point {
//variables
        ATTRIBUTE Real X;
        ATTRIBUTE Real Y;
//methods
        Float Distance (IN Point aPoint);
        //computes the distance between two points
        Float Equals (IN Point aPoint);
        //determines if two points have the same coordinates
}
```

- سنقوم باستخدام الصف Point في بناء الصف Rectangle الذي يحتوي على واصفتين من النمط Point (UpperLeftCorner و LowerRightCorner) بالإضافة إلى الطرائق (Area و Length و Height).

```
CLASS Rectangle {
                                //variables
        ATTRIBUTE Point UpperLeftCorner;
        ATTRIBUTE Point LowerRightCorner;
//methods
        Float Area ();
        //computes the area of the rectangle
        Float Length ();
        //compute the length
        Float Height ();
        //compute the height
}
```

علاقات الصفوف

- يتم تنظيم الصفوف في بنية هرمية، حيث يكون لكل صف أب وحيد.
- الصف الأعلى: هو عبارة عن تصنيف أعم للصفوف الجزئية منه.
- الصفوف الجزئية: تحتوي على مركبات مخصصة من التصنيف الأعم للصف الأعلى.
- الوراثة: هي قدرة الغرض في الهرمية على وراثة بنية المعطيات والسلوك للصفوف الأعلى منه في الهرمية، وهي نوعان: وراثة أحادية ووراثة متعددة.



يتم تنظيم الصفوف في بنية هرمية، حيث يكون لكل صف أب وحيد.

والصف الأعلى: هو عبارة عن تصنيف أعم للصفوف الجزئية منه.

أما الصفوف الجزئية: فتحتوي على مركبات مخصصة من التصنيف الأعم للصف الأعلى.

مثال: الصف (أداة موسيقية) هو صف أعلى للصفوف (بيانو، غيتار، فيولون)، وبالتالي فإن الصفوف الأخيرة هي صفوف جزئية من صف الأداة الموسيقية.

جميع الصفوف في الهرمية موروثة من الصف الجذر للهرمية.

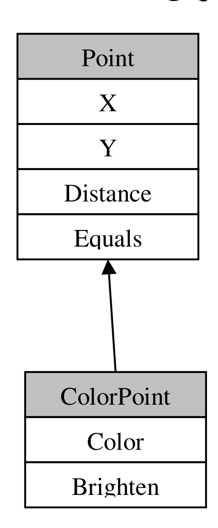
هنالك نوعان من الوراثة:

– وراثة أحادية: وهي موجودة عندما يكون للصف أب واحد فقط (صف أعلى)، وعندما يُرسِل النظام طلب تنفيذ طريقة معينة إلى غرض معين يتم البحث أولاً عن هذه الطريقة في الصف الذي ينتمي إليه الغرض ومن ثم في حال عدم وجودها يتم البحث في الصفوف الأعلى في الهرمية.

– وراثة متعددة: وهي موجودة عندما يكون للصف أكثر من أب واحد.

مثال 2

- نعود في هذا المثال إلى الصف Point وسنقوم بتطبيق مفهوم الوراثة لبناء صف جديد هو ColorPoint حيث يتضمن هذا الصف الجديد نفس واصفات وطرائق الصف Point، ولكن إضافة إليها الواصفة Color والطريقة Brighten.

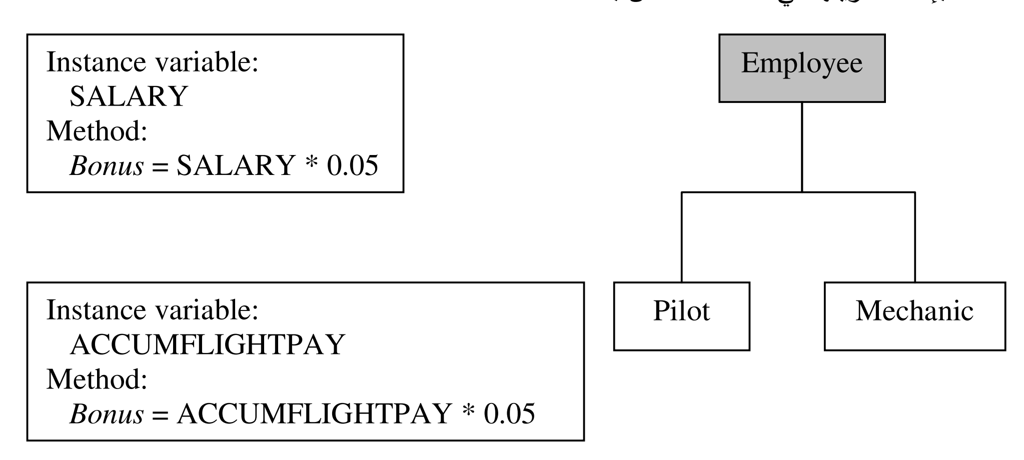


- تعريف الصف ColorPoint.

```
CLASS ColorPoint EXTENDS Point{
//variables
        ATTRIBUTE INTEGER Color;
        ATTRIBUTE Point LowerRightCorner;
//methods
        Integer Brighten ();
        //computes a new color that is brighter
}
```

إعادة تعريف الطرائق

- إعادة تعريف الطريقة: يمكن أن نعيد تعريف طريقة معرفة في الصف الأب، بإعادة تعريفها في الصفوف الجزئية منه.
- مثال: لدينا صف أب (موظف)، وصفوف جزئية منه (طيار وميكانيكي)، نلاحظ أنه أعدنا تعريف الطريقة (*Bonus*) في الصف *طيار* ولم نقم بإعادة تعريفها في الصف *ميكانيكي*، فجميع الموظفين لديهم طريقة واحدة في حساب المكافآت ماعدا الطيار لذلك قمنا بإعادة تعريفها في الصف الخاص به.

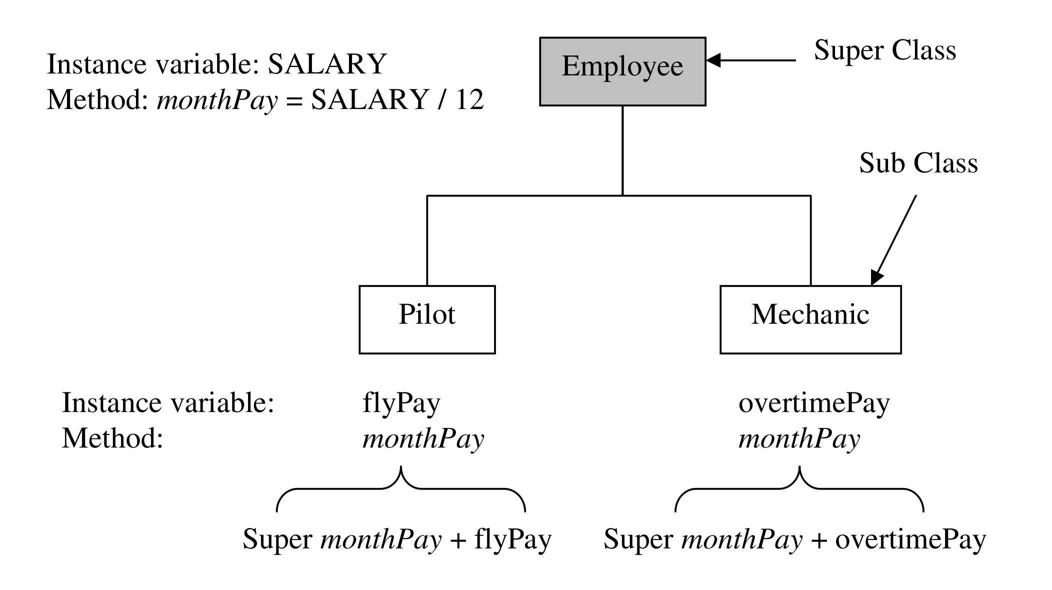


يُمكِن أن نعيد تعريف طريقة معرفة في الصف الأب، ضمن الصفوف الجزئية منه، وهذا ما يدعى إعادة التعريف.

في مثال إعادة التعريف لدينا صف أب (موظف)، وصفوف جزئية منه (طيار وميكانيكي)، نلاحظ أنه أعدنا تعريف الطريقة (*Bonus*) في الصف طيار ولم نقم بإعادة تعريفها في الصف ميكانيكي، فجميع الموظفين لديهم طريقة واحدة في حساب المكافآت ماعدا الطيار لذلك قمنا بإعادة تعريفها في الصف الخاص به.

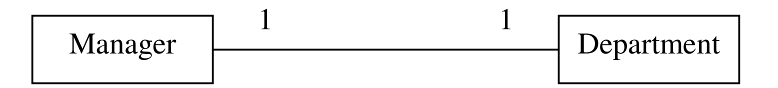
تعددية الأشكال

- تمكن تعددية الأشكال الغرض من السلوك بحسب معطياته الخاصة.
- مثال: بالعودة إلى نفس المثال السابق فإن حساب الراتب الشهري من خلال طلب نفس الطريقة (*monthPay*) من الصف *ميكانيكي* والصف *طيار* ولكن سيتم حسابها بطريقة مختلفة بكل صف وسيتم إعادة النتيجة الصحيحة.

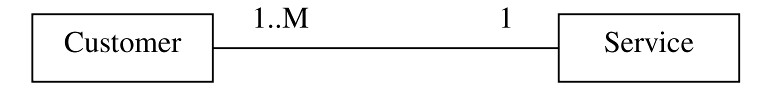


العلاقات بين الصفوف

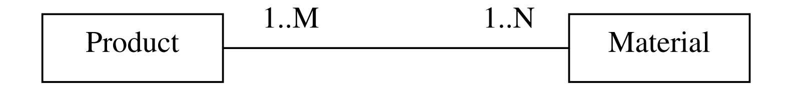
- علاقة واحد لواحد (1:1): علاقة غرض لغرض. مثال كل مدير يرأس قسم واحد وكل قسم يرأسه مدير واحد.



- علاقة واحد لكثير (1:M): كل غرض من الصف الأول يرتبط بـــM غرض من الصف الثاني. مثال الموظف والخدمة، يعمل الموظف على خدمة واحدة، بينما تحوي الخدمة أكثر من موظف (علاقة 1:M).



- علاقة كثير لكثير (M:N): يرتبط كل غرض من الصف الأول بـــM غرض من الصف الثاني، وكذلك يرتبط كل غرض من الصف الثاني بـــN غرض من الصف الأول. مثال كل منتج يحتاج إلى مجموعة مواد أولية، وكل مادة أولية تُستخدَم في إنتاج أكثر من منتج.



العلاقات بين الصف:
علاقة واحد لواحد (1:1): علاقة غرض لغرض. مثال كل مدير يرأس قسم واحد وكل قسم يرأسه مدير واحد.
علاقة واحد لكثير (1:M): كل غرض من الصف الأول يرتبط بـــM غرض من الصف الثاني. مثال الموظف والخدمة، يعمل الموظف على

خدمة واحدة، بينما تحوي الخدمة أكثر من موظف (علاقة 1:M).

علاقة كثير لكثير (M:N): يرتبط كل غرض من الصف الأول بـــM غرض من الصف الثاني، وكذلك يرتبط كل غرض من الصف الثاني بـــN غرض من الصف الأول. مثال كل منتج يحتاج إلى مجموعة مواد أولية، وكل مادة أولية تُستخدَم في إنتاج أكثر من منتج.

القسم الحادي عشر والثاني عشر والثالث عشر

مقدمة عن الصفوف والطرائق

الكلمات المفتاحية:

صف، طريقة، طريقة صف، طريقة نسخة.

ملخص:

يتعرف الطالب في هذا القسم على الصفوف وعلى طرائق الصفوف في مقاربة أولية تسمح بالتعرف على بعض الصفوف الجاهزة للإستخدام كسلاسل المحارف والجداول والمصفوفات.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- طريقة بناء برنامج C# مؤلف من صفّ واحد؛
- الإعلان عن طريقة تابعة للصف وتعريفها؛
- تمرير المُعاملات؛
- تعريف مدى المتحولات؛
- سلاسل المحارف؛
- الجداول والمصفوفات؛
- مسائل للحلّ.

يكفينا صفّ واحد !

يمكننا، وحسب ما لاحظنا في بعض الأمثلة التي وردت في الأقسام السابقة، أن بإمكان برنامج كامل مكتوب بلغة C# مؤلف من صف برمجي واحد أن يلعب دور برنامج رئيسي مكتوب بلغة برمجة خوارزمية أخرى مثل لغة C.

يبدأ مثل هذا الصف عادةً بالكلمة المفتاحية "class"، يليها إسم الصف وجسمه البرمجي المدعو كتلة الصف ويكون مُحاطاً بقوسين من الشكل "{" و "}".

يمكن لهذا الصف أن يتحول إلى صف متكامل قابل للعب دور برنامج رئيسي إذا احتوى بداخله طريقة (إجرائية تابعة للصف) تدعى "main"، تعمل على قدح عملية تشغيل البرنامج بكامله.

بعد أن يجري حفظ الصف في ملف "xxx.cs"، تولد عملية ترجمة الملف الآنف الذكر الملف "xxx.exe" الجاهز للتنفيذ.

أمثلة:

```
using System;
namespace Example
{
	/// <summary>
	///
	/// </summary>
	class Example1
	{
		static void Main(string[ ] args)
		{
			// You can add your program Here
		}
	}
}
```

```
using System;
namespace Example
{
	/// <summary>
	///
	/// </summary>
	class Example2
	{
		static void Main(string[ ] args)
		{
			Console.WriteLine("Good Morning !");
		}
	}
}
```

الطرائق ليست إلا إجرائيات !

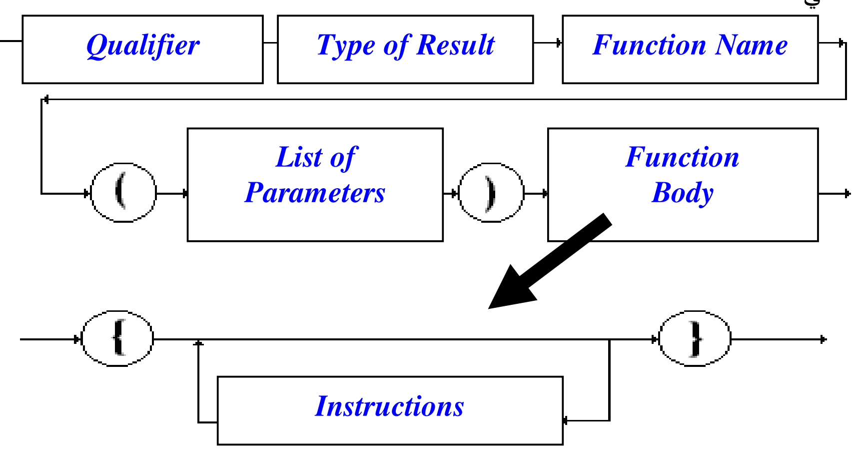
تمثل الطرائق، إجرائيات تُغلّف مجموعة من التعليمات التي تحدد طريقة عمل الصفّ. فبدون الطرائق، لايمكن للصف أن يقوم بأي عمل إلا احتواء عناصر تمثل متحولات الصفّ.

تميز C# نوعين من الطرائق: "**طريقة الصف**" نفسه والتي تظهر عند تعريف الصف، و"**نسخة الطريقة**" التابعة لغرض والناتجة عن نسخ طريقة الصف ضمن غرض (متحول) له نمط الصفّ.

سنكتفي في هذه المرحلة بطرائق الصف لتسهيل العمل. بالنتيجة، عندما نستخدم في الفقرات اللاحقة كلمة "**طريقة**" دون أية صفات أخرى لها، فإننا نعني بها "**طريقة صف**" وذلك لأن تطبيقاتنا لاتملك إلا صفاً واحداً بحيث يمكن اعتبار طرائق هذا الصف بمثابة إجرائيات التطبيق في حالة البرمجة العادية.

التصريح عن طريقة وتعريفها

يحتاج التصريح عن طريقة إلى تعريف رأس يحتوي على صورة عن المُعاملات اللازمة لعمل الطريقة، يجري بعدها تعريف جسم الطريقة الذي يحوي التعليمات التي سيجري تنفيذها عند استدعاء الطريقة. بشكل عام، يكون التصريح عن الطريقة وجسم الطريقة متتاليان بحسب الشكل القواعدي التالي:



برمجياً، يجري التصريح طريقة وفق الشكل القواعدي التالي:

<Qualifier><Type of Result><Function Name> (<Formal List of Parameters>)

وتكون دلالة هذا التصرح كما يلي:

- حتى الآن سنكتفي بالكلمة المفتاحية **static** ككلمة تعبر عن <Qualifier>، وهي كلمة تشير إلى أن الطريقة هي طريقة تابعة للصف في الصف المُعرَّفة فيه. ويمكن إهمال هذا الجزء من التصريح.
- تعيد الطريقة نتيجة تنتمي إلى أحد الأنماط البسيطة التي تعرفها #C، أو void (في حال لم يكن هناك نتائج إرجاع)، أو الأنماط المركبة التي يعرفها المُبرمج. ولايمكن إهمال هذا الجزء من التصريح.
- ويشبه التصريح عن المُعاملات، تعريف المتحولات والتصريح عنها، ويمكن لهذه اللائحة أن تكون فارغة.
- ويمكن لجسم الطريقة أن يكون فارغاً مُعبراً عن طريقة لاقيمة لها.

مثال:

```
using System;

class Application
{
	// Methods without parameters
	int compute1( ){
		//.....
	}

	bool test1( ){
		//.....
	}

	void recompute1( ){
		//.....
	}

	// Methods with parameters
	int compute2(byte a, byte b, int x ) {
		//.....
	}

	bool test2(int k) {
		//.....
	}

	void recompute2(int x, int y, int z ) {
		//.....
	}

	static void Main(string[ ] args) {
		//...
	}
}
```

يحتاج التصريح عن طريقة إلى تعريف رأس يحتوي على صورة عن المُعاملات اللازمة لعمل الطريقة، يجري بعدها تعريف جسم الطريقة الذي يحوي التعليمات التي سيجري تنفيذها عند استدعاء الطريقة. بشكل عام، يكون التصريح عن الطريقة وجسم الطريقة متتاليان.

تمرير المعاملات

- مقدمة -

تمثل المعاملات المُستخدَمة عند تعريف الإجرائيات، متحولات صماء.

تشبه هذه العملية، عملية تعريف تابع رياضي مثل f(x)=x+5

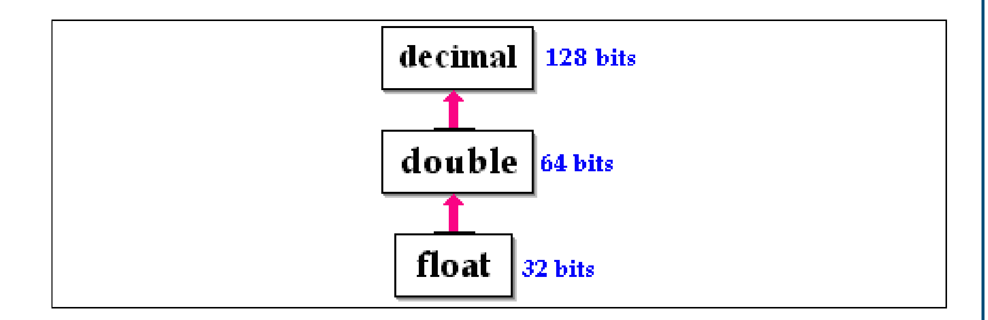
$$f(2)=7$$
$$f(7)=12$$
$$\ldots$$

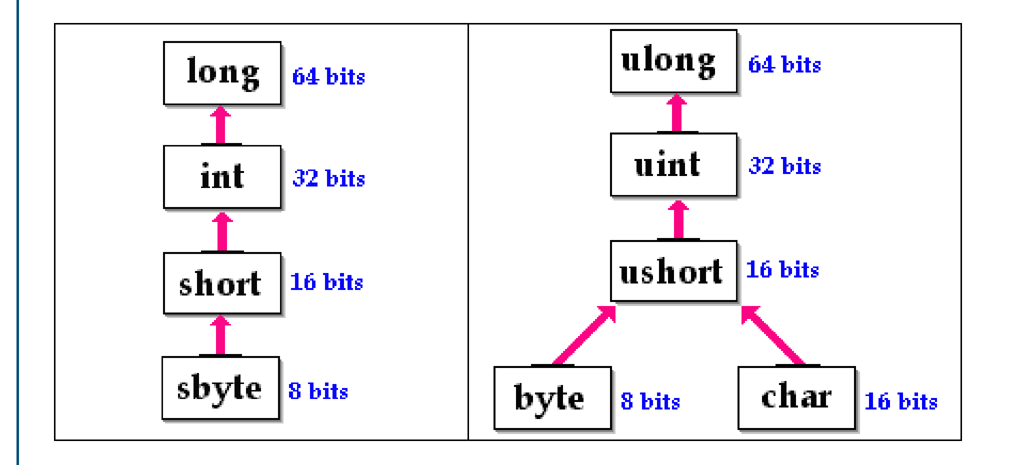
تمثل المعاملات المُستخدَمة عند تعريف الإجرائيات، متحولات صماء تساعد في تفسير عمل البرنامج من أجل متحولات حقيقية ستحل مستقبلاً (عند استدعاء الإجرائية وتنفيذها) محل هذه المُعاملات.

تشبه هذه العملية، عملية تعريف تابع رياضي مثل f(x)=x+5 حيث يلعب f هنا دور الإجرائية، وتلعب x دور المُعامل الأصمّ. تظهر المتحولات الحقيقية عند تنفيذ التابع، فعندما تحل القيمة 2 محل x يجري تنفيذ f فنحصل على القيمة 7.

تمرير المعاملات

- تجانس الأنماط البسيطة -





مثال:

```
using System;

class Application
{
	static void Main(string[ ] args)
	{
		int [ ] table= {12,-5,7,8,-6,6,4,78,2};
		sbyte elt = 4;
		short i ;

		for ( i = 0 ; i<8 ; i++ )
			if (elt==table[i]) break ;

		afficher(i,elt);
	}
	static void afficher (int ord , long val)
	{
		if (ord == 8)
		Console.WriteLine("Value : " + val + "Not Found.");
		else
		Console.WriteLine("Value : " + val + " Order :" + ord);
	}
}
```

يمكن، تبعاً لتسلسل الأنماط المتجانسة الظاهر في كل شكل من أشكال الشريحة أن يتمكن المتحول ذو النمط صاحب الحجم الأكبر أن يستوعب قيمةً تنتمي إلى نمط متجانس معه ذو حجم أصغر.

فعلى سبيل المثال، يمكن لمتحول من نمط short أن يحتوي قيمة (أن نسند له متحولاً) من نمط sbyte ولكن لايمكن أن يقبل قيمة من نمط char.

ينطبق هذا الكلام على مُعاملات، ففي حال تعريف إجرائية (طريقة) تمتلك معاملاً من النمط short مثلاً، يمكننا عندها استخدامها مع متحول طبيعي من نمط sbyte كبديل عن المُعامل.

تمرير المعاملات

– تمرير القيمة –

عند الإعلان عن الطرائق ومعاملاتها، وعند استدعائها، يكون أسلوب تمرير قيمة المتحولات صالحاً من أجل كافة المعاملات ذات الأنماط البسيطة في #C، بالإضافة إلى المعاملات التي تكون على شكل أغراض (أنماطاً مركبة).

عند استدعاء طريقة أو إجرائية تمتلك معامل ممرر بالقيمة، من أجل متحول ما، يقوم البرنامج ببناء نسخة من المتحول (نسخة من قيمته) وتمريرها إلى الإجرائية بحيث لايؤثر أي تعديل على النسخة المُمررة، على المتحول الأصلي.

تلقائياً يكون تمرير معاملات الطرائق، تمريراً للقيمة.

مثال:

```
using System;

class Application
{
	static void Main(string[ ] args)
	{
		int [ ] table= {12,-5,7,8,-6,6,4,78,2};
		int res = 0;

		short i ;
		for ( i = 0 ; i<8 ; i++ )
			res=Compute(i,res, table[i]);


		Console.WriteLine("Result : " + res);
	}

	static int Compute (short s, int r , int val)
	{
		int k=r+s*val;
		return k;
	}
}
```

عند الإعلان عن الطرائق ومعاملاتها، وعند استدعائها، يكون أسلوب تمرير قيمة المتحولات صالحاً من أجل كافة المعاملات ذات الأنماط البسيطة في #C، بالإضافة إلى المعاملات التي تكون على شكل أغراض (أنماطاً مركبة).

عند استدعاء طريقة أو إجرائية تمتلك معامل ممرر بالقيمة، من أجل متحول ما، يقوم البرنامج ببناء نسخة من المتحول (نسخة من قيمته) وتمريرها إلى الإجرائية بحيث لايؤثر أي تعديل على النسخة المُمررة، على المتحول الأصلي.

تلقائياً يكون تمرير معاملات الطرائق، تمريراً للقيمة.

تمرير المعاملات

– تمرير العنوان –

عند الإعلان عن الطرائق ومعاملاتها واستدعائها، يكون أسلوب تمرير عناوين المتحولات صالحاً من أجل كافة أنماط المعاملات في #C.

عند استدعاء طريقة أو إجرائية – تمتلك معامل ممرر بالعنوان– من أجل متحول ما، لايبني البرنامج نسخة عن المتحول، وإنما يستخدم المتحول نفسه، بحيث يؤثر أي تعديل على النسخة المُمررة، على المتحول الأصلي.

تأخذ عملية تعريف المُعامل المُمرر بالعنوان، وعملية التمرير بالعنوان، الشكل التالي:

```
static int mymethod (int a , ref char b)
{
 //.......
  return a+b;
}

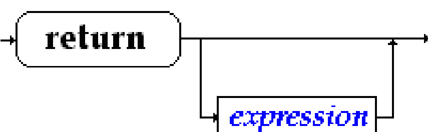
….

int x = 10, y = '$', z = 30;
z = mymethod(x, ref y) ;
```

بحيث يسبق المحول المُمرر بالعنوان كلمة ref المفتاحية.

إرجاع نتيجة طريقة

يمكن لأي طريقة أن تعيد قيمة من نمط محدد وذلك اعتماداً على الكلمة المفتاحية return التي تأخذ الشكل القواعدي التالي:



دلالياً، يجب أن تحقق return مايلي:

- يجب أن تعيد تعبير له نفس نمط الإرجاع المُعرَّف عند الإعلان عن الطريقة وتعريفها؛
- عند الوصول إلى return أثناء تنفيذ تعليمات الطريقة، يتوقف تنفيذ بقية التعليمات الموجودة ما بعد return؛
- عند وجود أكثر من مسار تنفيذي ممكن ضمن برنامج واحد (كوجود تعليمة if else، حيث تشكل if مسار تنفيذ لحالات معينة، وتشكل else مسار تنفيذ آخر لحالات أخر) يجب وضع تعليمة return في نهاية كل مسار تنفيذي. انظر المثال 2.

مثال1:

يحسب البرنامج التالي التابع f(x)=3x-7 من أجل x=4 ومن أجل x=5

```
using System;
class Application
{
        static void Main(string[ ] args)
        { // ...
                int x , y ;
                x = 4 ;
                y = f(5) ;
                y = f(x) ;

                Console.WriteLine("f(x)=" + f(x) );
                Console.WriteLine("f(5)=" + f(5) );
         }

         static int f (int x )
         {
                        return 3*x-7;
         }
}
```

مثال2:

تقوم الطريقة increment بإضافة 1 إلى قيمة المتحول إذا كان لايساوي الصفر.

```
using System;
class Application
{
        static void Main(string[ ] args)
        {
                int a = 0  ;
                a=Increment ( a );
                a=Increment ( a+4 );
        }

        static int Increment(int x)
        {
                if (x == 0)
                {
                        Console.WriteLine("The Case of 0 ...");
                        return x;
                }
                else
                {
                        Console.WriteLine("The other cases");
                        return x++;
                }
        }
}
```

مدى تعريف المتحولات

تقضي القاعدة الأساسية، بأن يكون المتحول مرئي (قابل للاستخدام) ضمن المقطع أو الكتلة التي جرى تعريفه فيها.

نعني بالمقاطع أو كتل التعليمات في لغة #C مايلي:

- الصفوف؛
- الطرائق؛
- التعليمات الأساسية المُركبة (تعليمة if else، أو تعليمة while، أو تعليمة do، أو تعليمة for).

بشكل عام:

- لايمكن تعريف متحول ضمن طريقة، إذا سبق وجرى تعريف معامل للطريقة أو متحول محلي للطريقة بنفس الإسم.
- لايمكن تعريف متحول ضمن مقطع (if، أو while، أو for، أو do)، إذا سبق وجرى تعريف متحول بنفس الإسم في أي مقطع يحتوي المقطع المذكور.

عناصر الصف، ومتحولات الطرائق

تكون المتحولات المُعرَّفة كعناصر ضمن الصف، قابلة للاستخدام من قبل جميع طرائق الصف. ويمكن أن تتبدل هذه القابلية للاستخدام من خلال استخدام كلمات مفتاحية من نمط public أو private لتعريف المتحول. سنتعرض لهذه الكلمات لاحقاً.

مثال1:

نلاحظ أن المتحول a هو أحد عناصر الصف، وهو مرئي في الطريقة g وفي الطريقة f. ففي الطريقة g يُستخدَم لحساب التعبير 3x-a أما في الطريقة f فيجري إخفاؤه بالمُعامل a الذي يُستَخدَم لحساب التعبير 3x-a.
(جرّب البرنامج ولاحظ النتيجة في حالتي f و g).

```
using System;
class Visiblity1
{
	static int a = 10;

	static void Main(string[ ] args)
	{
		int result_g, result_f;
		int x=10;

		result_g=g(x);
		result_f=f(x,5);

		Console.WriteLine("g(10)="+ result_g);
		Console.WriteLine("f(10,5)="+result_f);
	}

	static int g (int x )
	{
			return  3*x-a;
	}

	static int f (int x, int a )
	{
			return 3*x-a;
	}
}
```

ملاحظة:

تخضع عناصر الصف والمتحولات المُعرَّفة في طرائق الصف، إلى نفس القواعد الكلاسيكية المُستَخدَمة لتحديد مدى تعريف ورؤية المتحول

تمارين للتجريب

تمرين1:

اكتب برنامج لحساب مربعات الأعداد من 1 إلى 100 وذلك من خلال تعريف طريقة تدعى Square لحساب المُربَّع.

الحلّ:

بسيط جداً !!!!

تمرين2:

أعط نتيجة تنفيذ البرنامج التالي وعلل النتيجة التي تظهر في كل مرة وبعد كل استدعاء لطريقة من الطريقتين A و B.

```
using System;

class Scoping
{
        static int x=1;

        static void Main(string[ ] args)
        {
                Console.WriteLine("local x in method " + x);

                MethodA();
                Console.WriteLine("local x in 1st call of A: " + x);

                MethodB();
                Console.WriteLine("local x in 1st call of B: " + x);

                MethodA();
                Console.WriteLine("local x in 2nd call of A: " + x);

                MethodB();
                Console.WriteLine("local x in 2nd call of B: " + x);
        }

        static void MethodA()
        {
                int x = 25;
                ++x;
        }

        static void MethodB()
        {
                x *= 10;
        }
}
```

الحلّ: 1، 10، 10، 100.

تمرين 3:

نُعرِّف تابع العاملي بالشكل 1*2*3 .... *(x-2)*(x-1)*x!=x.

1. اكتب برنامج، يكرر الطلب من المُستخدم إدخال قيمة x من نمط ushort ومن ثم يستدعي طريقة Fact(x) بحيث تقوم هذه الطريقة بحساب x! (يستمر البرنامج بالعمل مع تكرار طلبات إدخال أرقام حتى يُدخل المُستخدِم الرقم 0). (مساعدة: استخدم لحساب Fact(x) إحدى تعليمات التكرار مثل while، أو do، أو for)
2. من أجل أي قيمة للمتحول x يبدأ البرنامج بإعطاء قيمة Fact(x) تساوي 0 ولماذا؟

الحلّ:

البرنامج المطلوب تنفيذه

```
using System;
class Factorial
{
	static int x=1;
	static void Main(string[ ] args)
	{
		string s;
		ushort a=1000;
		long f;

		s=Console.ReadLine();
		a=UInt16.Parse(s);

		while (a !=0)
		{
			f=Fact(a);

			Console.WriteLine("Fact of " + a + " = " + f);

			 s=Console.ReadLine();
			 a=UInt16.Parse(s);
		}
	}

	static long Fact(ushort a)
	{
		long res=a;

		while (a>1)
		{
			a--;
			res=res*a;
		}

		return res;
	}
}
```

تمرين 4:

نُعرِّف متتالية Fibonacci كمايلي:

$$u(n)=\begin{cases} n, \text{ if } n=0 \text{ or } n=1 \\ u(n-1)+u(n-2) \end{cases}$$

اكتب برنامج، يكرر الطلب من المُستخدم إدخال قيمة n من نمط ومن ثم يستدعي طريقة Fibonacci(n) بحيث تقوم هذه الطريقة بحساب قيمة u(n). (يستمر البرنامج بالعمل مع تكرار طلبات إدخال أرقام حتى يُدخل المُستخدِم عدداً سالباً).

الحلّ:

```
class Scoping
{
	static int x=1;
	static void Main(string[ ] args)
	{
		string s;
		int n;
		long f;

		s=Console.ReadLine();
		n=Int32.Parse(s);

		while (n >= 0)
		{
			f=Fibonacci(n);

			Console.WriteLine("U("+n+")=" + f);

			s=Console.ReadLine();
			n=Int32.Parse(s);
		}
	}
	static long Fibonacci( int number )
	{
		if ( number == 0 || number == 1 )
		return number;

		else
		return Fibonacci( number - 1 ) + Fibonacci( number - 2 );
	}
}
```

صفوف وبنى معطيات: سلاسل المحارف
– تعريف –

يُعتبر نمط المعطيات String (المُعبر عن سلسلة محارف) صفاً من صفوف فضاء الأسماء (المكتبة المرجعية) System، في إطار العمل DotNet.

بالتالي، لايمكن استخدام أي سلسلة محارف من نمط string إلا من خلال مجموعة الطرائق التابعة لهذا الصف.

صفوف وبنى معطيات: سلاسل المحارف
– التصريح عن سلسلة محارف –

يجري التصريح عن سلسلة محارف وفقاً لما يلي:

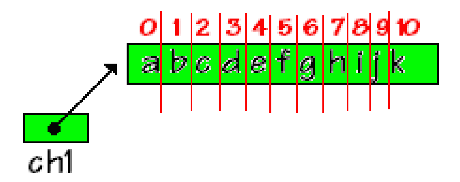
```
string ch1 ;
ch1 = "abcdefghijk";

string ch2 = "abcdefghijk";
```

صفوف وبنى معطيات: سلاسل المحارف

- التمثيل الداخلي لسلسلة محارف والوصول إلى مِحرف من محارف السلسلة -

يكون لسلسلة المحارف التمثيل الداخلي التالي:



يمكن الوصول إلى أحد محارف السلسلة باستخدام العملية ”[ ]“ (حيث تُقرأ السلسلة كجدول محارف)، وفقاً للمثال التالي:

```
string ch1 = "abcdefghijk";
char car = ch1[4] ; // contains the character 'e'
```

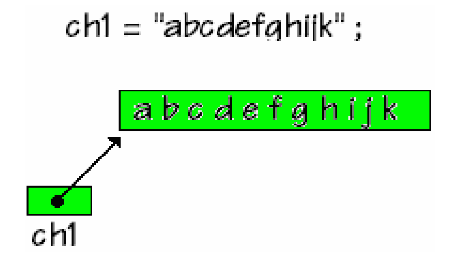
ولكن عملية الوصول تلك تقتصر على القراءة فقط، بحيث لايمكن تعديل محرف من محارف السلسلة باستخدام المؤشر [i]، وإنما اعتماداً على أدوات أخرى سنراها في الفقرات اللاحقة. فعلى سبيل المثال، إذ يعطي المترجم خطأً عند كتابة:

```
ch1[7]=car; // … Error
ch1[8]='x'; // … Error
```

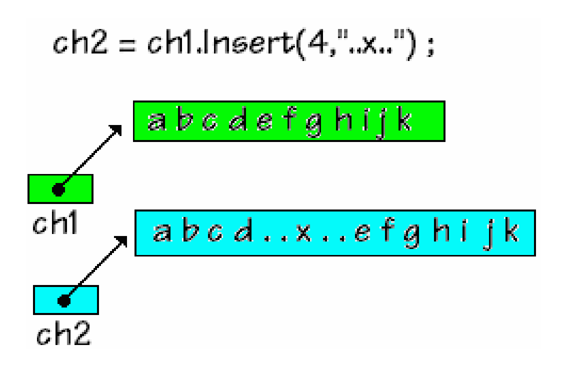
صفوف وبنى معطيات: سلاسل المحارف

- التعديل: الحشر Insert -

ليكن لدينا السلسلة ch1 التالية:



ولننفذ عليها عملية الحشر الظاهرة في الشكل، فنحصل على السلسلة ch2 الناتجة التالية:



يمتلك الصف string مجموعة من الطرائق، لحشر، أو نسخ، أو دمج، أو ... غيرها. ولاتسبب جميع هذه الطرائق تعديلاً في السلسلة نفسها وإنما تولد سلسلة جديدة ناتجة عن العملية.

صفوف وبنى معطيات: سلاسل المحارف

– التعديل: الدمج باستخدام عملية "+" –

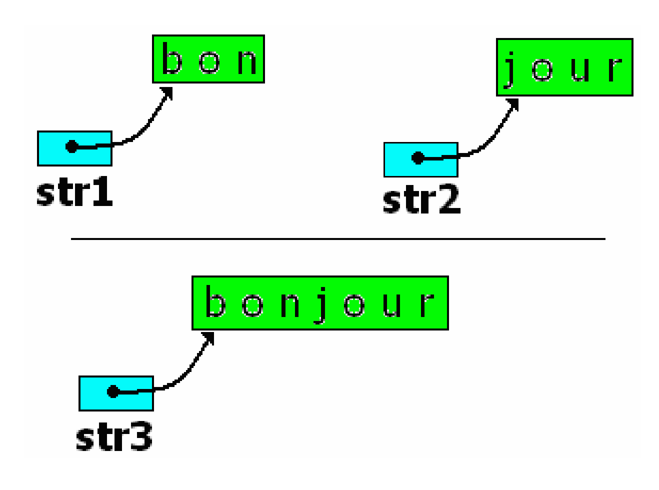
ليكن لدينا الوضع التالي:

```
string str1,str2,str3 ;

str1 = "bon" ;
str2 = "jour" ;

str3 = str1+str2 ;
```

نحصل بنتيجة الجمع (الدمج) على:



ونحصل على **طول السلسلة باستخدام التابع Length** كما يلي:

```
string str4 = "abcdef";
int Len;
Len = str1.Length ; // length = 6
```

يمتلك الصف string مجموعة من الطرائق، لحشر، أو نسخ، أو دمج، أو ... غيرها. ولاتسبب جميع هذه الطرائق تعديلاً في السلسلة نفسها وإنما تولد سلسلة جديدة ناتجة عن العملية.

صفوف وبنى معطيات: سلاسل المحارف

- التعديل: الحصول على موقع سلسلة جزئية من سلسلة محارف IndexOf -

يمكن أن نحصل على موقع بداية السلسلة الجزئية "cde" مثلاً ضمن السلسلة "abcde" باستخدام الطريقة IndexOf التي تعيد عدد صحيح برقم موقع أول ظهور للسلسلة الجزئية كمايلي:

```
string str5 = "abcdef" , ssch="cde";

int ord;

ord = str1.IndexOf ( ssch );
```

يمتلك الصف string مجموعة من الطرائق، لحشر، أو نسخ، أو دمج، أو ... غيرها. ولاتسبب جميع هذه الطرائق تعديلاً في السلسلة نفسها وإنما تولد سلسلة جديدة ناتجة عن العملية.

صفوف وبنى معطيات: سلاسل المحارف

- التعديل: تحويل سلسلة محارف إلى جدول محارف ToCharArray -

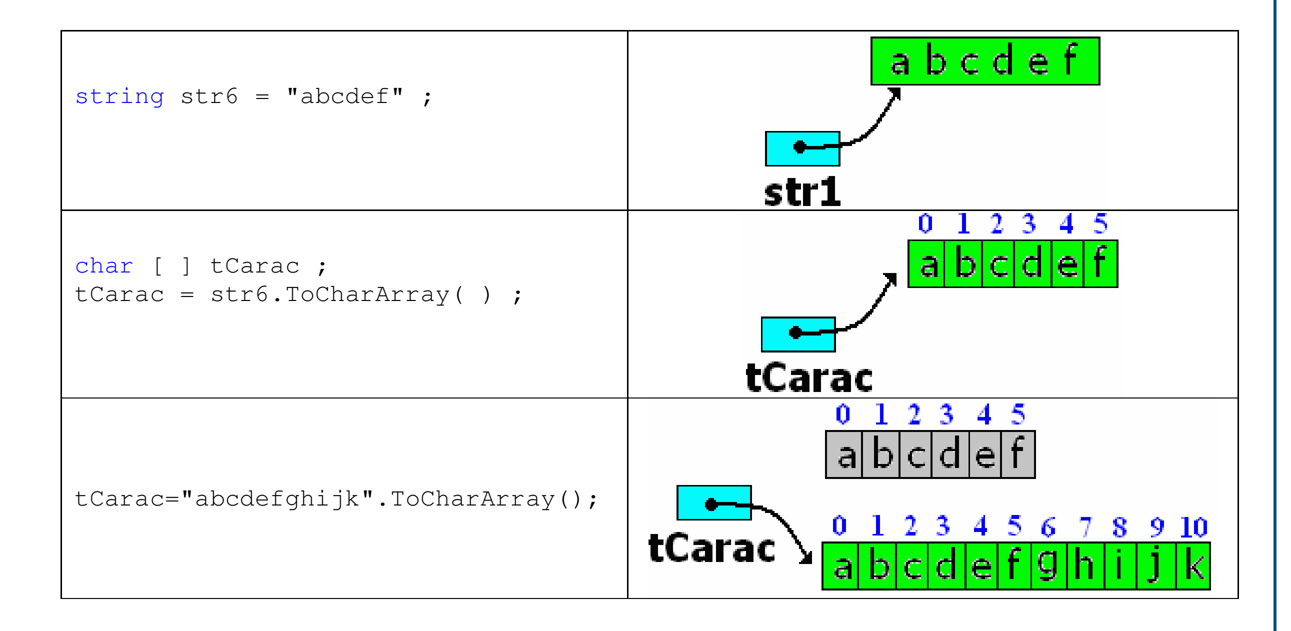
يمكن أن نرغب باستخدام سلسلة المحارف كجدول للمحارف لكي يتسنى لنا تطبيق عمليات خاصة بالجداول عليها. نستخدم في هذه الحالة الطريقة **ToCharArray** التي يمكن أن نستخدمها كمايلي:

```
string str6 = "abcdef" ;
char [ ] tCarac ;

tCarac = str6.ToCharArray( ) ;

tCarac = "abcdefghijk".ToCharArray( );
```

يوضح الجدول التالي التبديل الذي حصل مع تنفيذ كل تعليمة من التعليمات السابقة:



`string str6 = "abcdef" ;`	a b c d e f / str1
`char [ ] tCarac ;` `tCarac = str6.ToCharArray( ) ;`	0 1 2 3 4 5 / a b c d e f / tCarac
`tCarac="abcdefghijk".ToCharArray();`	0 1 2 3 4 5 / a b c d e f / tCarac / 0 1 2 3 4 5 6 7 8 9 10 / a b c d e f g h i j k

يمتلك الصف string مجموعة من الطرائق، لحشر، أو نسخ، أو دمج، أو ... غيرها. ولاتسبب جميع هذه الطرائق تعديلاً في السلسلة نفسها وإنما تولد سلسلة جديدة ناتجة عن العملية.

صفوف وبنى معطيات: سلاسل المحارف

– التعديل: الإسناد والمقارنة –

يمكن أن نستخدم عمليات الإسناد "=" مع سلاسل المحارف، كما يمكن أن نستخدم عملية مقارنة سلسلتين متساويتين "= =" أو باستخدام الطريقة "Equals". بالإضافة إلى ماسبق، يمكننا اعتباراً من محتوى سلسلة أن نبني سلسلة جديدة مساوية لها باستخدام تعليمة "new"

وهي عملية مختلفة عن عملية الإسناد حسب ما سنوضحه فيما يلي:

```
string s1,s2,s3,s4, ch;

ch = "abcdef";
s1 = ch;
s2 = "abcdef";
s3 = new string("abcdef".ToCharArray( ));
s4 = new string(s3.ToCharArray( ));

Console.WriteLine("s1="+s1);
Console.WriteLine("s2="+s2);
Console.WriteLine("s3="+s3);
Console.WriteLine("s4="+s4);
Console.WriteLine("ch="+ch);

if( s2 == ch )Console.WriteLine("s2==ch");
else Console.WriteLine("s2<>ch");

if( s2 == s3 )Console.WriteLine("s2==s3");
else Console.WriteLine("s2<>s3");

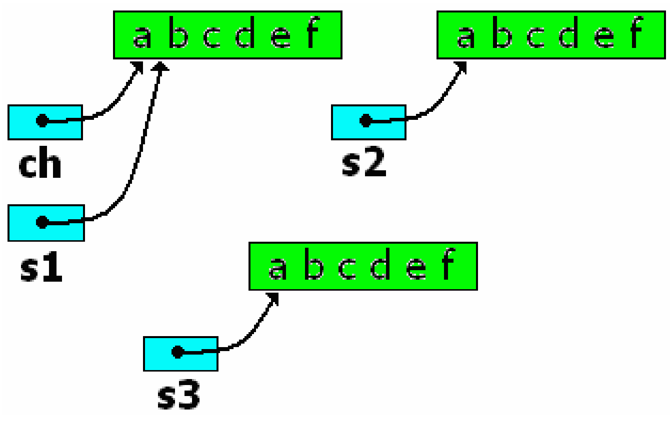
if( s3 == s4 )Console.WriteLine("s3==s4");
else Console.WriteLine("s3<>s4");

if( s3 == ch )Console.WriteLine("s3==ch");
else Console.WriteLine("s3<>ch");

if( s3.Equals(ch) )Console.WriteLine("s3==ch");
else Console.WriteLine("s3<>ch");

if( s3.Equals(s4) )Console.WriteLine("s3==s4");
else Console.WriteLine("s4<>ch");
```

يكون الفرق بين عملية الإسناد (حالة ch و s1) وعملية نسخ المحتوى (حالة s3 و s4 موضحاً فيما يلي:



ويعطي تنفيذ البرنامج الخرج التالي:

```
D:\MyProject\MyFirstAppl
s1=abcdef
s2=abcdef
s3=abcdef
s4=abcdef
ch=abcdef
s2==ch
s2==s3
s3==s4
s3==ch
s3==ch
s3==s4
```

يمتلك الصف string مجموعة من الطرائق، لحشر، أو نسخ، أو دمج، أو ... غيرها. ولاتسبب جميع هذه الطرائق تعديلاً في السلسلة نفسها وإنما تولد سلسلة جديدة ناتجة عن العملية.

صفوف وبنى معطيات: الجداول والمصفوفات

- تعريف جدول -

تعريف جدول بدون تحديد حجمه:

```
int [ ] table1;    // Table of integre
char [ ] table2;   // Table of char
float [ ] table3; // Table of float
string [ ] tableStr;// Table of string
```

تعريف جدول مع تحديد حجمه:

```
int [ ] table1 = new int [5];
char [ ] table2  = new char [12];
float [ ] table3  = new float [8];
string [ ] tableStr  = new String [9];
```

حيث تشير new إلى بناء غرض جديد من النمط المُحدد (int، أو char أو غيره) بعدد خانات مُحدد بالرقم الموضوع ضمن قوسين.

تعريف جدول مع إعطائه قيماً ابتدائية مباشرةً:

```
int [ ] table1 = {17,-9,4,3,57};
char [ ] table2  = {'a','j','k','m','z'};
float [ ] table3  = {-15.7f, 75, -22.03f, 3 ,57 };
string [ ] tableStr  = {"cat","dog","mouse","cow"};
```

صفوف وبنى معطيات: الجداول والمصفوفات

– تعريف مصفوفة (جدول متعدد الأبعاد) –

تعريف جدول بدون تحديد حجمها:

```
int [ , ] table1;
char [ , ] table2;
float [ , ] table3;
string [ , ] tableStr;
```

تعريف مصفوفة مع تحديد حجمها:

```
int [ , ] table1 = new int [5, 2]; // 5 Lines x 2 Columns
char [ , ] table2 = new char [9,4]; // 9 Lines x 4 Columns
float [ , ] table3 = new float [2,8]; // 2 Lines x 8 Columns
string [ , ] tableStr = new String [3,9]; // 3 Lines x 9 Columns
```

حيث تشير new إلى بناء غرض جديد من النمط المُحدد (int، أو char أو غيره) بعدد خانات مُحدد بالرقمين الموضوعين ضمن قوسين واللذين يحددان عدد خانات المصفوفة طولاً وعرضاً.

توليد خانات مصفوفة لم تتحدد أبعادها عند التعريف:

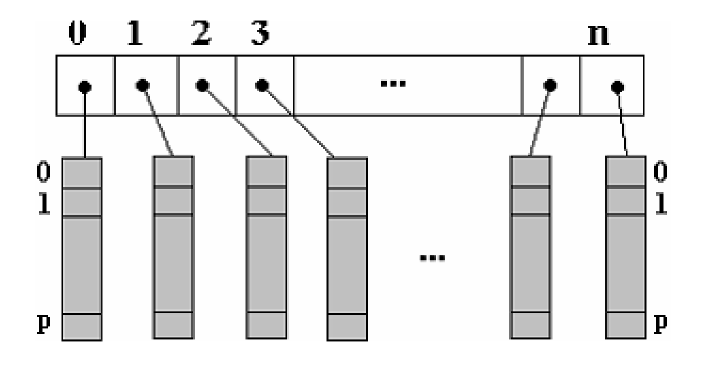
يجري توليد خانات مصفوفة لم تتحدد أبعادها وتهيأتها يدوياً وفقاً للمثال التالي والذي نريد فيه توليد خانات المصفوفة t[n+1][p+1]:

```
int n=10, p=8;

int [ ][ ] table = new int [n+1][ ];

for (int i=0; i<n+1; i++)
      table[i] = new int [p+1];
```

ويمكن تمثيل مثل هذه المصفوفة بالشكل:



صفوف وبنى معطيات: الجداول والمصفوفات

– استخدام الجداول والمصفوفات –

يمكن تنفيذ عمليات اسناد على الجداول والمصفوفات واستخدامها ضمن تعليمات مختلفة مع الإنتباه إلى أن الجدول الذي طوله n، تكون خاناته موزعة بين الخانة رقم 0 والخانة رقم n-1.

أمثلة:

```
int [ ] table1 = new int [5];

// Assignment Operations:
table1[0] = -458;
table1[4] = 5891;
table1[5] = 72;  // Error table1[5] does not exists

// Loop:
for (int i = 0 ; i<= table1.Length-1;  i++)
      table1[i] = 3*i-1;
// Result: table1 = {-1,2,5,8,11}
```

```
char [ ] table2  = new char [7];

// Assignment Operations:
table2[0] = '?' ;
table2[4] = 'a' ;
table2[14] = '#' ; // Error table1[5] does not exists

//Loop
for (int i = 0 ; i<= table2.Length-1;  i++)
      table2[i] =(char)('a'+i);
// Result:  table2 = {'a', 'b', 'c' ,'d', 'e', 'f'}
```

مسائل للحلّ خوارزمياً، ومن ثم بلغة #C

مسألة 1:

نريد كتابة برنامج لحلّ معادلة من الدرجة الثانية من الشكل $Ax^2 + Bx + c = 0$. (فكرّ بالخوارزمية أولاً ومن ثم اكتب البرنامج بعد وضعها بلغة شبه التشفير)

الحلّ:

```
using System;
namespace Equation
{
        class Application2DEquation
        {
                static void Main (string[ ] arg)
                {
                        double a, b, c, delta ;
                        double x, x1, x2 ;
                        Console.Write("Entrer une valeur pour a : ") ;
                        a = Double.Parse( System.Console.ReadLine( ) ) ;
                        Console.Write("Entrer une valeur pour b : ") ;
                        b = Double.Parse( System.Console.ReadLine( ) ) ;
                        Console.Write("Entrer une valeur pour c : ") ;
                        c = Double.Parse( System.Console.ReadLine( ) ) ;

                        if (a ==0)
                        {
                                if (b ==0)
                                {
                                        if (c ==0)
                                        {
                                                Console.WriteLine("Each Real is a solution") ;
                                        }
                                        else
                                        {// c ≠ 0
                                                Console.WriteLine("No Solution") ;
                                        }
                                }
                                else
                                { // b ≠ 0
                                        x = -c/b ;
                                        Console.WriteLine("The Solution is : " + x) ;
                                }
                        }
                        else
                        { // a ≠ 0
                                delta = b*b - 4*a*c ;
                                if (delta < 0)
                                {
                                Console.WriteLine("No Solution in the set of real numbers") ;
                                }
                                else
                                { // delta ≥ 0
                                x1 = (-b + Math.Sqrt(delta))/ (2*a) ;
                                x2 = (-b - Math.Sqrt(delta))/ (2*a) ;
                                Console.WriteLine("Two solutions equals to " + x1 + " et " + x2) ;
                                }
                        }
                }
        }
}
```

مسألة 2:

ندعو عدد Armstrong، كل عدد يكون مساوياً لحاصل جمع مكعبات الأرقام التي تؤلفه. مثال:

$153 = 1^3 + 5^3 + 3^3 = 1 + 125 + 27$.

هناك عدة أعداد Armstrong وكلها من مرتبة المئات، اكتب برنامج بلغة C# لتحديدها. (فكرّ بالخوارزمية أولاً ومن ثم اكتب البرنامج بعد وضعها بلغة شبه التشفير)

الحلّ:

```
using System;
namespace Armestrong
{
        class ApplicationArmstrong
        {
                static void Main(string[ ] args)
                {
                        int i, j, k, n, sumcube;
                        Console.WriteLine("Number of Armstrong:");

                        for(i = 1; i<=9; i++)
                                for(j = 0; j<=9; j++)
                                        for(k = 0; k<=9; k++)
                                        {
                                                n = 100*i + 10*j + k;
                                                sumcube = i*i*i + j*j*j + k*k*k;

                                                if (sumcube == n)
                                                        Console.WriteLine(n);
                                        }
                }
        }
}
```

مسألة 3:

نقول عن عدد أنه perfect إذا كان يساوي حاصل جمع جميع قواسمه بما فيهم الواحد. اكتب برنامج بلغة C# يأخذ عدد n ويبحث عن أول n عدد Perfect من مجموعة الأعداد الطبيعية. (فكرّ بالخوارزمية أولاً ومن ثم اكتب البرنامج بعد وضعها بلغة شبه التشفير)

الحلّ:

```
using System;
namespace Perfect
{
        class ApplicationPerfect
        {
                static void Main (string[ ] args)
                {
                        int compt = 0, n, k, sumdiv, nbr;
                        Console.Write("Number of Perfect numbers you wish find : ");

                        n = Int32.Parse( Console.ReadLine( ) ) ;
                        nbr = 2;

                        while (compt != n)
                        {
                                sumdiv = 1;
                                k = 2;

                                while(k <= nbr/2 )
                                {
                                        if (nbr % k == 0)
                                            sumdiv += k ;

                                        k++;
                                }

                                if (sumdiv == nbr)
                                {
                                        Console.WriteLine(nbr+" is a Perfect number");
                                        compt++;
                                }
                                nbr++;
                        }
                }
        }
}
```

مسألة 4:

اكتب برنامج بلغة C# يقرأ جدولاً من n رقم ويرتبها بالترتيب التصاعدي أو التنازلي حسب خوارزمية الترتيب بالفقاعات Bubble Sort.

الخوارزمية التي يجب فهمها وتحويلها إلى برنامج بلغة #C:

```
Algorithm Bubble_Sort;

local: i , j , n, temp: Natural Numbers;
Input-Output : Tab : Table of n Integers;

Begin

for i from n until 1 Do

   for j from 2 until i Do

      if Tab[ j-1 ] > Tab[ j ] then
         temp=Tab[ j-1 ] ;
         Tab[ j-1 ]= Tab[ j ] ;
         Tab[ j ] = temp ;
      End_if

   End_for

End_for

End_Bubble_Sort
```

مسألة 5:

اكتب برنامج بلغة #C يقرأ جدولاً من n رقم ويرتبها بالترتيب التصاعدي أو التنازلي حسب خوارزمية الترتييب بالحشر Sort by Insertion.

الخوارزمية التي يجب فهمها وتحويلها إلى برنامج بلغة #C:

```
Algorithm Sort_by_Insertion;

local: i , j , n, v: Natural Numbers;
Input-Output : Tab : Table of n Integers;

Begin

for i from 2 until n do

    v = Tab[ i ] ;
    j= i ;

    while (Tab[ j-1 ]> v)
        Tab[ j ] =Tab[ j-1 ];
        j = j-1;
    End_While ;

    Tab[ j ] =v ;

End_for

End_Sort_by_Insertion
```

القسم الرابع عشر والخامس عشر

المبادئ الأساسية للبرمجة بلغة غرضية التوجه

الكلمات المفتاحية:

غرض، مِثل/نسخة، فضاء أسماء

ملخص:

يهدف هذا الفصل إلى تعريف الطالب بلغة #C كلغة برمجة غرضية التوجه تعتمد على تعريف بنى مُركبة على شكل صفوف، وعلى فكرة وراثة هذه الصفوف لبعضها، وعلى عملية تغليف هذه الصفوف دون الدخول في تفاصيل معقدة عن هذا الموضوع الذي سنستعرضه بالتفصيل في مواد لاحقة تختص بالبرمجة غرضية التوجه.

أهداف تعليمية:

يتعرف الطالب في هذا الفصل على:

- الصف كنمط مُركَّب؛
- الأغراض؛
- إنشاء الأغراض وتدميرها؛
- طرائق البناء،
- الوراثة؛
- التغليف؛
- إعادة تعريف الطريقة.

.

تحديد مدى تعريف ورؤية المتحولات

public	تكون العناصر المسبوقة بكلمة public قابلة للاستخدام من قبل جميع الصفوف والإجرائيات الأخرى.
private	تكون العناصر المسبوقة بكلمة private قابلة للاستخدام من قبل عناصر الصف الذي تتعرف فيه فقط.
protected	تكون العناصر المسبوقة بكلمة protected قابلة للاستخدام من قبل عناصر الصف الذي تتعرف فيه ومن قبل الصفوف والعناصر المُشتقة منه والتي ترثه (سنتعرض لعملية الوراثة لاحقاً).

نستخدم في جميع اللغات الغرضية التوجه، كلمات مفتاحية تساعد على تعريف مدى رؤية متحول، وتحديد حالات استخدامه من قبل طرائق وإجرائيات أخرى. يوضح الجدول التالي هذه الحالات:

- تكون العناصر المسبوقة بكلمة public قابلة للاستخدام من قبل جميع الصفوف والإجرائيات الأخرى.
- تكون العناصر المسبوقة بكلمة private قابلة للاستخدام من قبل عناصر الصف الذي تتعرف فيه فقط.
- تكون العناصر المسبوقة بكلمة protected قابلة للاستخدام من قبل عناصر الصف الذي تتعرف فيه ومن قبل الصفوف والعناصر المُشتقة منه والتي ترثه (سنتعرض لعملية الوراثة لاحقاً).

الصفوف: أنماط مُركَّبة غرضية التوجّه

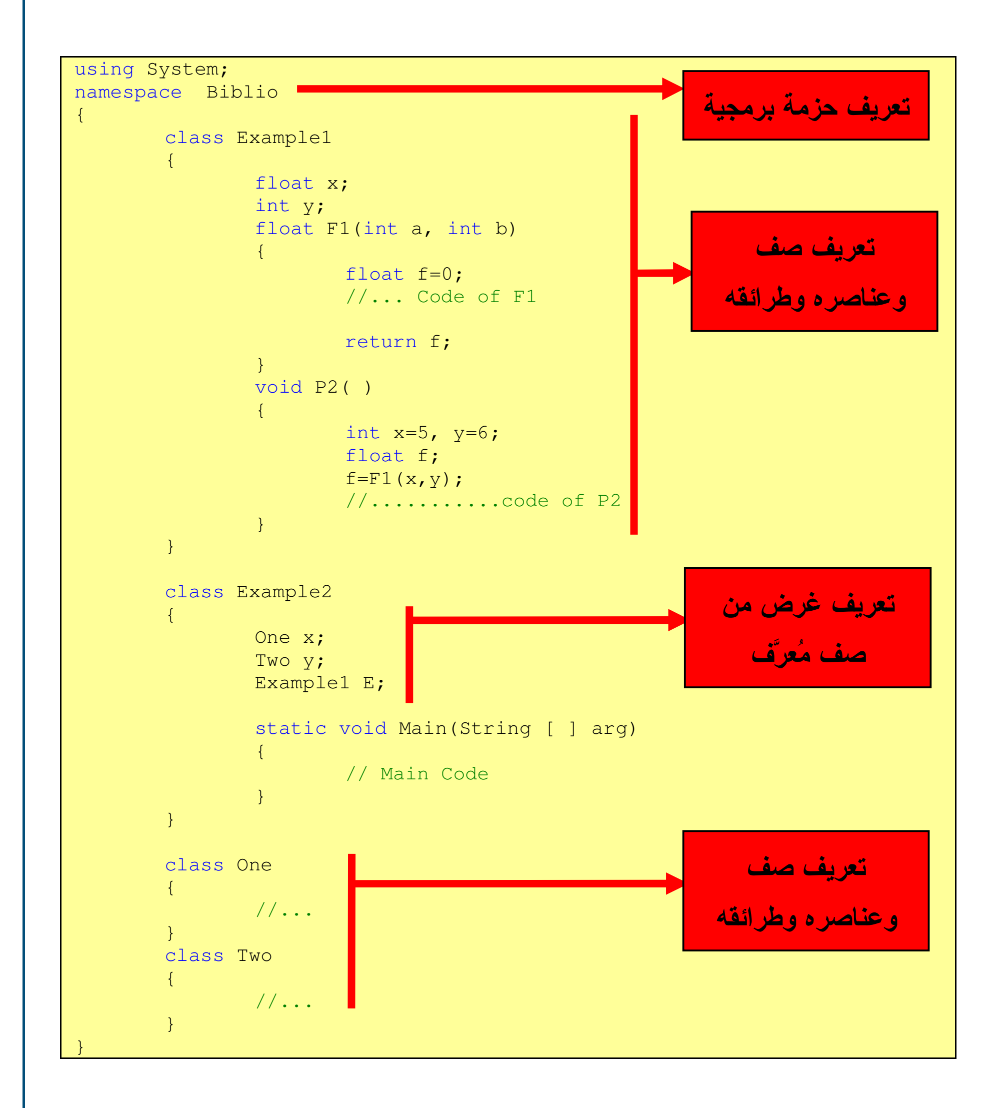
يمتلك أي برنامج C# صفاً واحداً على الأقل، ويمكن أن يكون مسبوقاً بتصريح لاستخدام صفوف أخرى مُعرَّفة مُسبقاً (اعتماداً على الكلمة المفتاحية using) وموجودة ضمن حزمة (مكتبة) برمجية.

يتجلى تعريف الحزمة البرمجية الحاوية على مجموعة صفوف قابلة لإعادة الاستخدام، باستخدام الكلمة المفتاحية namespace التي تصرّح عن تعريف حزمة برمجية.

في C#، يكون تعريف وتنجيز الصف موجوداً في مكان واحد وهو مكان تعريف الحزمة البرمجية التي يؤلف الصف أحد عناصرها. وأي صف، لاتظهر أمامه إحدى الكلمات المفتاحية المُستخدمة في تحديد مدى الرؤية والتعريف، يُعتبر صفاً عاماً (public).

يتألف الصف من مجموعة من المتحولات التي ندعوها عناصر الصف أو أعضاء الصف، ومجموعة من الإجرائيات التي ندعوها طرائق الصف.

التصريح عن صفّ وتعريف أغراض منه


```
using System;
namespace  Biblio
{
        class Example1
        {
                float x;
                int y;
                float F1(int a, int b)
                {
                        float f=0;
                        //... Code of F1

                        return f;
                }
                void P2( )
                {
                        int x=5, y=6;
                        float f;
                        f=F1(x,y);
                        //...........code of P2
                }
        }

        class Example2
        {
                One x;
                Two y;
                Example1 E;

                static void Main(String [ ] arg)
                {
                        // Main Code
                }
        }

        class One
        {
                //...
        }
        class Two
        {
                //...
        }
}
```

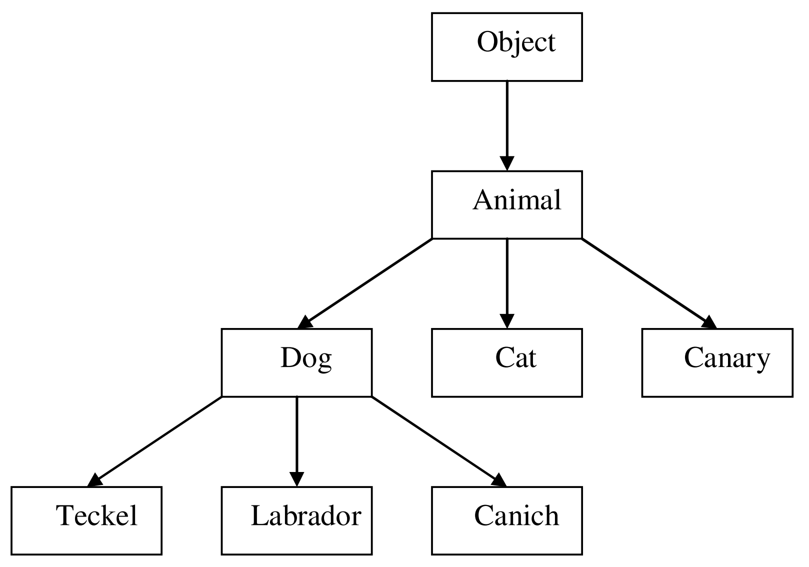

الأغراض

- تتعامل لغة #C مع كل العناصر المُستَخدَمة كأغراض؛
- الغرض هو مِثِل أونسخة من صف؛
- كل صف مُشتق من صف أب؛

- يكون أي صف مُعرَّف بدون تحديد سلفه، صفاً خلفاً للصف Object.

مثال:

لنأخذ المثال التالي التي يعبر عن هرمية بين الحيوانات:



إن الهرمية الموضحة في المثال هي هرمية (is a) وليست هرمية (has a)، فكل Teckel هو كلب (Dog)، وكل كلب هو حيوان (Animal)، وكل حيوان هو كائن أو غرض (Object)، فنستنتج أن كل Teckel هو حيوان، وأن كل Teckel هو أيضاً غرض.

تتعامل لغة #C مع كل العناصر المُستَخدَمة كأغراض، حتى المتحولات الأولية يمكن معالجتها كأغراض بعد تغليفها بالصفوف المُناسبة.

والغرض في #C هو مثِل أونسخة من صف، وكل صف مُشتق من صف في مستو أعلى، ونسميه أحيناً الصف الأب، والصف الوحيد المُستثنى هو الصف Object الذي يُعتبر السلف الأقدم بين الصفوف. ويكون أي صف مُعرَّف بدون تحديد سلفه، صفاً خلفاً للصف Object.

من ناحية أخرى، يكون مثل من صف هو غرض من النمط الموافق، ومن أنماط كل الصفوف الأسلاف.

البنّاء

- طرائق خاصة تحمل أسماء الصفوف التي تنتمي إليها؛
- يجري تنفيذها تلقائياً عند إنشاء الغرض؛
- يمكن للصف الواحد أن يحتوي أكثر من بنّاء، يكون لكل منها مُعاملات مختلفة.

مثال:

ليكن لدينا الصف التالي:

```
class one {
      int a;
}
```

تقوم #C أوتوماتيكياً بتوليد بنّاء خاص بهذا الصف هو:

```
public one() { }
```

لذا، يمكن عند تعريف الصف، تعريف بنّاء له:

```
class one {
      int a;
      public one() { }
}
```

بحيث يمكن الاستفادة منه في عملية إعداد عناصر الصف:

```
class one {
      int a;
      public one() {
            a=100;
      }
}
```

كما يمكن تعريف البنّاء مع معاملات:

```
class one {
      int a;
      public one(int b) {
                  a=b;
      }
}
```

ومن الممكن تعريف أكثر من بنّاء للصف الواحد تختلف باختلاف معاملاتها:

```
class one {
      int a;

      public one() {
      }

      public one(int b) {
                  a=b;
      }

      public one(short b) {
                  a=b;
      }
}
```

تكون الطرائق التي ندعوها البنّاء، طرائق خاصة تحمل أسماء الصفوف التي تنتمي إليها ويجري تنفيذها تلقائياً عند إنشاء الغرض (عند تعريف غرض من الصف). بمعنى آخر يهدف البنّاء إلى تأطير عملية توليد غرض من الصف.

يمكن للصف الواحد أن يحتوي أكثر من بنّاء، يكون لكل منها مُعاملات مختلفة.

وفي حال لم يجر الإعلان عن بنّاء لصف مكتوب بلغة C# تقوم اللغة أوتوماتيكياً بتوليد بنّاء للصف.

لاأهمية للبنّاء إلا إذا كان مرئياً لكافة الصفوف الأخرى التي تريد بناء نسخ من الصف المعني، لذا يجري وضعالكلمة المفتاحية public أمام الإعلان عن البنّاء.

إنشاء الأغراض

يتطلب إنشاء المثل من صف تنفيذ طريقة بنّاء الصف من خلال استخدام الكلمة المحجوزة new على أن يليها اسم طريقة بناء الصف، ثم قائمة الوسطاء إن وجدت.

مثال:

ليكن لدينا الصف التالي:

```
class Employee
{
        int register;
        static int number;

        public Employee()
        {
                register = ++number;
        }

        public void showRegister()
        {
                Console.WriteLine("The employee register =" + register);
        }
}
```

هنا لابد من التساؤل عن كيفية إنشاء غرض من الصف Employee وكيف يمكن استخدامه. للإجابة على هذا التساؤل، سنجرب البرنامج التالي:

```
public class Test
{
        static void Main(string[] args)
        {
                Employee E=new Employee();
        }
}
```

حيث نلاحظ أننا أضفنا الصف Test الذي يتضمن الطريقة Main التي ستمكننا من تشغيل البرنامج. يقوم هذا البرنامج بإنشاء مثل من الصف Employee. يمكننا اعتباراً من هذه اللحظة استخدام المثل كما هو واضح فيما يلي:

```
public class test
{
        static void Main(string[] args)
        {
                Employee E=new Employee();

                E.showRegister();
        }
}
```

يتطلب إنشاء المثل من صف تنفيذ طريقة بنّاء الصف من خلال استخدام الكلمة المحجوزة new على أن يليها اسم طريقة بناء الصف، ثم قائمة الوسطاء إن وجدت.

تدمير الأغراض: مُجمِّع النفايات (Garbage Collector)

تلقي بعض لغات البرمجة مسؤولية تحرير مواقع الذاكرة التي تشغلها أغراض غير مفيدة على عاتق المبرمج، ويعتبر ذلك مصدراً أساسياً من مصادر الأخطاء التي تحدث عند تشغيل البرامج إذ كثيراً ماتبقى أجزاء هامة من ذاكرة الآلة محجوزة لأغراض انتهت فترة حياتها واستخدامها، وتعتبر عملية ملاحقة هذه الأغراض وتحرير مواقعها من المهام الصعبة، في حين تبدو بالمقابل النتائج المترتبة على بقائها غير ذات أهمية سيما عندما تكون حجوم هذه الأغراض صغيرة.

إلا أن المشكلة تبدأ بالظهور مع تراكم هذه الأجزاء مما يؤدي لحدوث مانسميه بظاهرة تسرب الذاكرة (memory leaks)، وهي تؤثر بالدرجة الأولى على البرنامج الأكثر استخداماً، أي نظام التشغيل.
ولتقدير أهمية المشكلة، تصور برنامجاً مؤلفاً من عدة ملايين من الأسطر مع ماتتضمنه من إنشاء بنى معطيات يبقى قسماً منها موجوداً في الذاكرة بعد انتفاء الحاجة لوجودها. لذا نحتاج في كثير من الأحيان إلى إعادة إقلاع الحاسب لتحرير تلك المناطق. وقد نصادف بعض المشاكل مع بعض البرامج التطبيقية كبرامج معالجة النصوص، إلا أنه من المرجح أن مصممي هذه البرامج يتوقعون أن الغالبية العظمى من المستخدمين تغلق البرنامج وتوقف الحاسب بعد نهاية يوم العمل لتعيد تشغيله من جديد في اليوم التالي مما يؤدي بالنتيجة إلى تحرير مناطق من الذاكرة.

لقد حلت #C هذه المشكلة بطريقة بسيطة للغاية، وذلك بواسطة برنامج نسميه مُجمِّع النفايات يعمل تلقائياً عندما يصبح حجم الذاكرة المتاحة أقل من حد معين وبذلك نضمن عدم بقاء أغراض غير ضرورية في الذاكرة إلى الحد الذي يؤدي إلى ظهور مشكلة.

تسمح هذه التقنية بتجنب قسم كبير من المشاكل لكن ليس كلها، فمعظم البرامج الحديثة ذات طبيعة خاطبية، أي تنتظر في الكثير من الأحيان الحصول على إجابات من المستخدم، وبالتالي يستطيع مُجمِّع النفايات أن يعمل في الخفاء ودون لفت انتباه المستخدم فهو يستطيع استثمار الفترات الفاصلة بين عمليات النقر على الفأرة أو على مفاتيح لوحة المفاتيح.

بالمقابل تحتاج بعض البرامج للعمل في الزمن الحقيقي وإبطاء عمل مثل تلك البرامج قد يطرح مشكلات جدية، لذا لاتعتبر لغة #C مناسبة لمثل هذه التطبيقات، فمُجمِّع النفايات إجراء غير متزامن أي لايمكن ضبط إيقاع عمله مع الإجراءات الأخرى.

الوراثة

```
class Animal
        {
                bool alive;
                int age;

                void getold()
                {
                        ++age;
                }

                void die()
                {
                        alive=false;
                }

                void cry()
                {
                        // ...
                }
        }
```

```
class Dog: Animal
{
        // ... Definition of Dogs inherited from Animal
}

class Canary: Animal
{
        // ... Definition of Canary inherited from Animal
}
```

عندما نقول عن كلب (Dog) إنه حيوان (Animal) (مع التأكيد على أنها صفوف لبرنامج C# وليست أغراضاً حقيقية) فإن هذا القول يشير ضمناً إلى أن الكلب يرث كل الصفات المميزة للحيوان، كذلك الأمر يرث الكلب من نوع Teckel كل الصفات المميزة للكلب، بل حتى كل الصفات المميزة للحيوان.

يكون لكل حيوان حالة (الصفة حي التي يمكن أن تكون صح أو خطأ، والعمر الذي يكون قيمة عددية)، كما يستطيع الصف أن يقوم بعدة أفعال (( )cry، أو ( )die، أو ( )getold، وهي الطرق). نستنتج مباشرةً أن لكل كلب من نوع Caniche، ولكل طائر Canary الصفات المميزة نفسها.

التغليف

تغليف الصف	التوصيف
abstract	لايمكن إنشاء أمثال أونسخ من الصف المميز بكلمة abstract. يفيد هذا الصف كأب لمجموعة صفوف تشترك من خلال وراثته بصفات مميزة مشتركة. **ملاحظة:** يمكن للطرائق أيضاً أن تكون abstract وتكون دون أي جسم أو تنجيز وتُستَخدَم كتواقيع لطرائق يمكن أن نقوم بتعريف عملها في الصفوف التي ترث الصف الذي يحتوي الطريقة من النوع abstract.
private	يكون الصف المميز بكلمة private مرئياً (قابلاً للاستخدام) من قبل جميع الصفوف المُعرَّفة في نفس فضاء الأسماء الذي جرى تعريفه فيه.
protected	يكون الصف المميز بكلمة protected مرئياً (قابلاً للاستخدام) من قبل جميع الصفوف التي ترث الصف الحاوي عليه.
public	يكون الصف المميز بكلمة public مرئياً (قابلاً للاستخدام) من قبل أي برنامج ينتمي لنفس فضاء الأسماء أو ينتمي لفضاء أسماء آخر.

مثال:

```
namespace Exemple
{
	public class ApplicationClasses
	{
		abstract class ApplicationClasse1 {
			// ...
		}

		public class ApplicationClasse2 {
			// ...
		}

		protected class ApplicationClasse3 {
			// ...
		}

		private class ApplicationClasse4 {
			// ...
		}

		class ApplicationClasse5 {
			// ...
		}

		class ApplicationTestClasses1 {
			ApplicationClasses.ApplicationClasse2 a2;
		}

		class ApplicationTestClasses2 {
			ApplicationClasse2 a2;
			ApplicationClasse5 a5;
		}

		class ApplicationTestClasses3 {
			ApplicationClasse1 a1;
			ApplicationClasse2 a2;
			ApplicationClasse2 a3;
			ApplicationClasse4 a4;
			ApplicationClasse5 a5;
		}

		static void Main(string[] args)
		{
			//...
		}
	}
}
```

يمكن في C# أن يجري تعريف صف ضمن صف آخر لذا تتطبق قواعد تعريف مدى الرؤيا ومجال التعريف على الصفوف أيضاً وليس على المتحولات والطرائق فقط.

يمكن تحديد مدى رؤية الصف ومجال تعريفه ضمن فضاء الأسماء، اعتماداً على نفس الكلمات المفتاحية. فإذا لم يجر وضع أي كلمة مفتاحية أمام تعريفه، يكون الصف مرئياً في كامل فضاء الأسماء الذي جرى تعريفه فيه.

هناك 4 كلمات مفتاحية للدلالة على مجال تعريف الصف: abstract، وpublic، وprivate، وprotected. وبدون استخدام أحد هذه المُعرِّفات يكون الصف من النوع public.

أما ضمن الصف، فتكون جميع الأعضاء والطرائق التي لاتسبقها إحدى الكلمات المفتاحية السابقة، من النوع private.

الرجوع إلى الصف الأب وإعادة تعريف الطرق

لنعد إلى مثال الحيوانات والوراثة ولنكتب الصف Animal مع بنّاؤه، ولننشئ في البرنامج الرئيسي مثل من هذا الغرض:

```
using System;
namespace Animals
{
	class Animal
	{
		bool alive;
		int age;

		public Animal()
		{
			// ...
		}
		public Animal (int a)
		{
			age = a;
			alive = true;

			Console.WriteLine("A "+age+" years old Animal is created");
		}
		public void getold()
		{
			++age;
		}
		public void die()
		{
			alive=false;
		}
		public void cry()
		{
			// ...
		}
	}

	public class test
	{
		static void Main(string[] args)
		{
			Animal A=new Animal(3);
		}
	}
}
```

يمكننا الآن البدء بتعريف الصفوف المشتقة من Animal، ولنبدأ بالصف Canary حيث يشير مايلي إلى أنه صف مشتق من Animal:

```
class Canary: Animal
{
		// ...
}
```

يحتاج الصف الجديد إلى طريقة بناء تأخذ كوسيط عدداً صحيحاً يمثل عمر الكناري في حال أردنا توليد كناري بعمر معين.

إن مايميز الكناري عن أي حيوان آخر وفق نموذجنا هو صوته فقط، ولبان الفرق لابد لنا من استخدام تقنية جديدة دعوها إعادة تعريف الطرق.

يمتلك الصف Canary بحكم الوراثة كل الحقول الأعضاء في الصف Animal، ومنها متحولات الأمثال والطرائق، ويمكن استخدام تلك المتحولات، وإسناد قيم جديدة لها كمايمكن استخدام طرق الصف الأب، لابل حتى إعادة تعريفها. وهذا ما سنقوم به بالنسبة للطريقة

.cry()

```
class Canary : Animal
{
	public Canary (int a)
	{
		new Animal(3);
	}
	public void cry ()
	{
		Console.WriteLine("Cui-Cui!");
	}
}
```

ويمكن عندها كتابة الصف test كمايلي:

```
public class test
{
	static void Main(string[] args)
	{
		Canary C=new Canary(2);
		C.cry();
	}
}
```

مسألة

اكتب برنامج Square يحفظ طول ضلعه ويمتلك طريقة لحساب مساحته. اشتق منه صف Cube الذي يمتلك طريقة تعريف لمساحة المكعب.

الحلّ:

```
using System;

namespace Redefinition
{
	class square
	{
		public int length;

		public square() {}

		public square(int len)
		{
			length=len;
		}

		public int getSurface()
		{
			return length*length;
		}
	}

	class cube:square
	{
		public cube(int len)
		{
			length=len;
		}

		public int getSurface()
		{
			return 6*length*length;
		}

		public static void Main()
		{
			cube c=new cube(3);
			square car=new square(3);
			Console.WriteLine("Surface of square car(3): ",car.getSurface());
			Console.WriteLine("Surface of cube c(3) :",c.getSurface());
		}
	}
}
```

مسألة

اكتب صفاً Point لتمثي نقطة يمتلك عنصرين x و y من النمط double يمثلان إحداثيات النقطة. واكتب صفاً آخر Line نحتاج لتعريفه إلى إحداثيات نقطتين: الأولى x1 و y1 والثانية x2 و y2 يجري طلبهما من المستخدم في البرنامج الرئيسي Main، ويقدم طريقة تعرض طول الخط المُعرَّف اعتباراً من النقطتين السابقتين

الحلّ:

```
using System;
namespace Redefinition
{
	class Point
	{
		public double Px,Py;
		public Point() {}
		public Point(double x, double y)
		{
			Px=x;
			Py=y;
		}
	}
	class Line
	{
		Point P1;
		Point P2;
		public Line(double x1, double y1, double x2, double y2)
		{
			P1=new Point(x1,y1);
			P2=new Point(x2,y2);
		}
		public double getLength()
		{
			double lenX, lenY;
			return Math.Sqrt(((P1.Px-P2.Px)*(P1.Px-P2.Px))+((P1.Py-P2.Py)*(P1.Py-P2.Py)));
		}
		public static void Main()
		{
			double x1,x2,y1,y2;
			string s;
			Console.Write("First Point X1 : ");
			x1=double.Parse(Console.ReadLine());
			Console.Write("First Point Y1 : ");
			y1=double.Parse(Console.ReadLine());
			Console.Write("Second Point X2 : ");
			x2=double.Parse(Console.ReadLine());
			Console.Write("Second Point Y2 : ");
			y2=double.Parse(Console.ReadLine());
			Line L=new Line(x1,y1,x2,y2);
			Console.WriteLine("Length of L : " + L.getLength());
			Console.ReadLine();
		}
	}
}
```

القسم السادس عشر والسابع عشر والثامن عشر

تمارين ومسائل للمناقشة والحلّ

ملخص:

يهدف هذا القسم إلى تقديم مجموعة من التمارين والمسائل حول مجموعة من الخوارزميات الأساسية التي ينبغي فهمها وحلها وتطبيقها بلغة C#.

أهداف تعليمية:

يتعامل الطالب في هذا الفصل مع مجموعة من التمارين التطبيقية التي تركز على ما تعلمه خلال الفصول السابقة من المادة.

تمارين ومسائل للمناقشة والحلّ

التمرين الأول:

اكتب بلغة #C برنامجاً لحساب القاسم المشترك الأعظم لعددين صحيحين لايساويان الصفر. اقترح الخوارزمية المناسبة لتنفيذ العمل، واكتبها بلغة شبه التشفير قبل المباشرة بكتابة البرنامج بلغة #C.

الحلّ1:

```
using System;
namespace Exercice1
{
        class ApplicationEuclide
        {
                static void Main (string[ ] args)
                {
                        Console.Write("First Number : ");
                        int p = Int32.Parse( Console.ReadLine( ) ) ;

                        Console.Write("Second Number : ");
                        int q = Int32.Parse( System.Console.ReadLine( ) ) ;

                        if (p*q!=0)
                                Console.WriteLine("mgcd of "+p+" and "+q+" is "+mgcd(p,q));
                        else
                                Console.WriteLine("One of the numbers is null !");
                }

                static int mgcd (int a , int b)
                {
                        int r,t ;
                        if ( b>a)
                        {
                                t = a;
                                a = b;
                                b = t;
                        }
                        do
                        {
                                r = a % b;
                                a = b;
                                b = r;
                        } while(r !=0);
                        return a ;
                }
        }
}
```

الحلّ2:

```
using System;
namespace Exercice1
{
	class ApplicationEgyptien
	{
		static void Main (string[ ] args)
		{
			Console.Write("First Number : ");
			int p = Int32.Parse( Console.ReadLine( ) ) ;
			Console.Write("Second Number : ");
			int q = Int32.Parse( Console.ReadLine( ) ) ;

			if ( p*q != 0 )
				System.Console.WriteLine("mgcd of "+p+" and "+q+" is "+mgcd(p,q));
			else
				Console.WriteLine("One of the numbers is null !");
		}

		static int mgcd (int p, int q)
		{
			while ( p != q)
			{
				if (p > q) p -= q;
				else q -= p;
			}
			return p;
		}
	}
}
```

التمرين الثاني:

اكتب بلغة #C برنامجاً لإظهار أول n عدد أولي من مجموعة الأعداد الصحيحة الموجبة. اقترح الخوارزمية المناسبة لتنفيذ العمل، واكتبها بلغة شبه التشفير قبل المباشرة بكتابة البرنامج بلغة #C.

الحلّ:

```
using System;
namespace Exercice2
{
	class ApplicationPrem
	{
		static void Main(string[ ] args)
		{
			int divis, nbr, n, count = 0 ;
			bool is_prem;

			Console.Write("How much numbers to Display ? ");
			n = Int32.Parse( Console.ReadLine( ) ) ;

			Console.WriteLine( 2 );
			nbr = 3;
			while (count < n-1)
			{
				divis = 2 ;
				is_prem = true;
				do
				{
					if (nbr % divis == 0) is_prem=false;
					else divis = divis+1 ;
				}
				while ((divis <= nbr/2) && (is_prem == true));
				if (is_prem)
				{
					count++;
					Console.WriteLine( nbr );
				}
				nbr++ ;
			}
		}
	}
}
```

التمرين الثالث:

اكتب بلغة C# برنامجاً للتحقق من أن سلسلة محارف تمتلك صفة PALINDROME، أي أنها تبقى نفسها سواء قرأناها من اليمين إلى اليسار أو من اليسار إلى اليمين.

مثال: ab,cddc,ba

اقترح الخوارزمية المناسبة لتنفيذ العمل، واكتبها بلغة شبه التشفير قبل المباشرة بكتابة البرنامج بلغة C#.

الحلّ:

```
using System;
namespace Exercice3
{
        class palindrome
        {
                static string convert ( string s )
                {
                        char [ ] tChar = s.ToCharArray ( );
                        char car ;
                        for( int i = 0 , j = tChar.Length - 1 ; i < j ; i ++ , j -- )
                        {
                                car = tChar[i] ;
                                tChar[i] = tChar[j] ;
                                tChar[j] = car ;
                        }
                        return new string ( tChar );
                }

                static void Main ( string [ ] args )
                {
                        Console.WriteLine ("Your string :");
                        string Mystring = Console.ReadLine ( );
                        string strCnv = convert ( Mystring );
                        if( Mystring == strCnv )
                                Console.WriteLine ("palindrome !");
                        else
                                System .Console.WriteLine ("Not palindrome !");

                        Console.ReadLine ( );
                }
        }
}
```

التمرين الرابع:

اكتب بلغة C# برنامجاً لتحويل تاريخ مكتوب بشكل رقمي إلى تاريخ مكتوب بصيغة حقيقية. نفترض في هذا البرنامج أن للأيام أرقام: الأحد=1، الإثنين=2، ...، السبت=7. كما نفترض أن للأشهر أرقام مُتعارف عليها (مثال: الشهر الرابع هو شهر نيسان). بالنتيجة تكون صيغة الدخل الرقمية هي (من اليسار إلى اليمين):

2/25/4/2006

وهي تعني:

Monday 25 April 2006

مساعدة للحلّ:

يمكن إيجاد الحل باستخدام الأنماط **enum** و **string**.

التمرين الخامس:

اكتب بلغة #C برنامجاً يقوم بعملية بحث خطي تسلسلي عن عنصر x ضمن جدول T مؤلف من n عنصر ويعطي ترتيبه في حال وجوده.

اقترح الخوارزمية المناسبة لتنفيذ العمل، واكتبها بلغة شبه التشفير قبل المباشرة بكتابة البرنامج بلغة #C.

الحلّ: سنقترح هنا الخوارزمية فقط بلغة التشفير ونترك للطالب تطبيقها كبرنامج.

```
i ←1;

while (i < n) and (T[i] <> x) do
        i ←i+1;
end_while

if T[i] = x then
        ord←i;
        write("the element exists, order:",ord);
else
        write("element not found")
end_if
```

التمرين السادس:

ليكن لدينا جدول T مرتب ترتيباً تصاعدياً ويحتوي على N عنصر، وليكن x عنصر من هذا الجدول. اشرح هدف وعمل الخوارزمية التالية، وطبقها كبرنامج بلغة #C.

```
Bottom, Middle, Top, Order : Integer;

Bottom←1;
Top←N;
Order←-1 ;

repeat
   Middle←(Bottom + Top) div 2;
   if x = T[Middle] then
     Order←Middle;
   else
      if T[Middle] < x then
         Bottom←Middle + 1;
      else
          Top←Middle-1;
      end_if
   end_if
until ( x = T[Middle] )
```

قراءات اضافية

- http://www.cacs.louisiana.edu/~mgr/404/burks/pcinfo/progdocs/
- http://eric_rollins.home.mindspring.com/introProgramming/
- http://www.softsteel.co.uk/tutorials/cSharp/contents.html
- http://www.functionx.com/csharp/index.htm
- http://www.csharphelp.com/
- http://www.ssw.uni-linz.ac.at/Teaching/Lectures/CSharp/Tutorial/
- C# How to Program, Dietel & Associates, Prentice Hall, 2003